А. Вуд

МИКРО-ПРОЦЕССОРЫ в вопросах и ответах

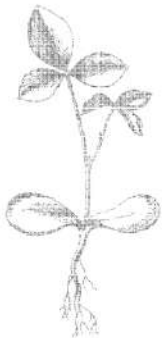
А. Вуд

МИКРО-ПРОЦЕССОРЫ в вопросах и ответах

Перевод с английского
Л.П. ЗАХАРОВОЙ

Под редакцией
Д.А. ПОСПЕЛОВА

Москва
Энергоатомиздат
1985



ББК 32.973
В 88
УДК 681.32—181.48

Рецензент И. В. Прангишвили

ALEC WOOD
MICROPROCESSORS: YOUR QUESTIONS ANSWERED
BUTTERWORTHS & Co.
Sevenoaks, 1982

Вуд А.

В 88 Микропроцессоры в вопросах и ответах/Пер. с англ. под ред. Д. А. Поспелова. — М.: Энергоатомиздат. 1985. — 184 с., ил.

70 к. 80000 экз.

В книге известного английского специалиста в форме вопросов и ответов даны основные сведения по цифровой технике. Изложены логические и арифметические основы ЭВМ. Рассмотрены логическое и программное обеспечение ЭВМ, аппаратные средства, структура и работа памяти, устройства ввода-вывода на базе микропроцессоров.

Для широкого круга инженеров и техников, самостоятельно изучающих современную вычислительную технику на базе микропроцессоров.

В $\frac{2405000000\text{-}029}{051(01)\text{-}85}$ 262-85

ББК 32.973
6Ф7

Алек Вуд

МИКРОПРОЦЕССОРЫ В ВОПРОСАХ И ОТВЕТАХ

Редактор издательства А. Н. Гусяцкая
Художественный редактор Т. А. Дворецкова
Обложка художника В. Я. Батищева
Технический редактор Н. П. Собакина
Корректор З. Б. Драновская

ИБ № 999

Сдано в набор 21.09.84. Подписано в печать 25.12.84. Формат 84×108$^1/_{32}$. Бумага кн.-журн. имп. Гарнитура литературная. Печать высокая. Усл. печ. л. 9,66. Усл. кр.-отт. 9,92. Уч.-изд. л. 10,37. Тираж 80 000 экз. Заказ № 979. Цена 70 к.

Энергоатомиздат, 113114, Москва, М-114, Шлюзовая наб., 10

Владимирская типография Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли 600000, г. Владимир, Октябрьский проспект, д. 7

Предисловие редактора перевода

Слово «микропроцессор» сейчас так же популярно, как было популярным два десятилетия назад слово «алгоритм». Микропроцессоры внесли в развитие вычислительной техники качественно новые свойства. Только благодаря им стали возможны карманные калькуляторы, персональные вычислительные машины, размещаемые в небольших корпусах размером с электрическую пишущую машинку, но равные по мощности огромным ЭВМ десятилетней давности; только с помощью микропроцессоров, по-видимому, удастся внедрить вычислительную технику во все отрасли народного хозяйства, только на них возлагают надежду разработчики систем управления интеллектуальными роботами. И, наконец, самое главное. Только переход к микропроцессорам позволил приблизить вычислительные машины к непрограммирующим пользователям и сделать реальной перспективу, когда вычислительная техника станет настолько же привычной в быту, насколько стали привычными для всех стиральные машины или телевизоры.

Но это означает, что область, известная ранее только специалистам в области вычислительной техники и программирования, становится нужной самым широким кругам, начиная от специалистов по управлению и кончая домашними хозяйками. А это означает, что информация о том, как устроены и работают микропроцессоры и как их использовать в различных областях, должна быть доступной для тех, кто традиционно не относился к группе специалистов в области вычислительных машин и методов их применения.

Среди существующих публикаций по микропроцессорам практически нет книги, в которой в популярной форме излагалась бы работа таких устройств.

Книга, которую Вы держите в руках, как раз и рассчитана на тех, кто мало знаком с особенностями вычислительных машин, но хотел бы быть в курсе современных тенденций их развития. Ее автор, А. Вуд, долгие годы занимался разработкой микропроцессоров и способст-

вовал их внедрению в ряд областей. Своей книгой, как это следует из аннотации, имеющейся в английском оригинале, он хотел бы «развеять мистический туман, которым окружен микропроцессор». И надо отдать должное А. Вуду. Во многом это ему удалось. Язык, которым написана книга, конечно, не совсем прост. От читателя требуется желание разобраться в написанном. Пытливый читатель найдет в этой небольшой книге ответы на массу вопросов, которые возникают у всех, кто впервые входит в не совсем привычный мир микроэлектроники и микрокомпьютеров. Перевод книги снабжен достаточным количеством примечаний, позволяющих читателю лучше понять терминологию, используемую автором, и суть излагаемых им сведений.

В какой-то мере книга А. Вуда может оказаться полезной и для тех, кто уже искушен в области микропроцессоров. Как во всякой бурно развивающейся области техники, в микропроцессорной технике все время возникают новые понятия и термины. И поэтому переводчик книги в максимальной степени постарался, сохранив оригинальное английское написание этих новых понятий, помочь не только новичку, но и специалисту уловить смысл терминов, встречаемых в журнальных публикациях по микропроцессорам.

У некоторых читателей может возникнуть недоумение, почему автор слишком большое внимание уделяет аппаратной части микропроцессоров в ущерб программированию. Развитие вычислительной техники последних десятилетий имело явную тенденцию к отделению пользователя от аппаратной реализации программ. Вместо реальных физических устройств пользователь все чаще имел дело с виртуальной машиной, концептуальной моделью базы данных и т. п., которые существовали лишь в воображении пользователя. Но особенность микропроцессоров и состоит в том, что в них впервые аппаратная реализация большинства пользовательских функций стала эффективной и возможной из-за резкого удешевления аппаратуры. Поэтому пользователь вновь приблизился к той физической структуре вычислительной машины, от которой он неуклонно отдалялся, когда развивались традиционные средства вычислительной техники.

Автор книги совсем не говорит о тех тенденциях использования микропроцессоров, которые сегодня становятся определяющими. Это прежде всего создание на их

основе распределенных вычислительных систем, способных в асинхронном режиме параллельно выполнять совокупность процедур, связанных, например, с обработкой больших массивов информации, получаемой от объекта управления. Так же перспективно использование микропроцессоров в интеллектуальных полуавтономных и автономных роботах, которые будут использоваться в гибких автоматизированных производствах и для работы в тех средах, в которых человеку пока трудно работать, и, наконец, в домашних компьютерах, которые в большом количестве появятся в ближайшие годы в составе самых разнообразных изделий на наших рабочих местах и в наших квартирах.

Поэтому так важно знать о микропроцессорах как можно больше. И книга Вуда — первый шаг на пути к этим знаниям.

В книге в основном сохранены обозначения автора. Для понимания схем в приложении 3 дана таблица соответствия обозначений автора и по действующим ГОСТ.

Переводчик и редактор выражают признательность всем, кто своими советами и замечаниями помог улучшить перевод этой книги. В частности, несколько ценных исправлений внес в текст Л. Я. Розенблюм. Особую благодарность хотелось бы выразить А. Л. Дудко, внимательно прочитавшему всю рукопись перевода и сделавшему полезные замечания и исправления, многие из которых нашли свое отражение не только в тексте, но и в примечаниях редактора и переводчика.

Д. А. Поспелов

Предисловие автора

Со времени передачи программы Би Би Си «Горизонт» для микросхем наступила пора заката*. Самым модным словом сейчас стало слово «микропроцессор». Однако проникновение в мир микропроцессоров часто сопряжено с трудностями из-за постоянного возникновения новых идей, появления новых терминов, понятных лишь профессионалам.

Цель этой небольшой книги — ответить на возможно большее число вопросов, возникающих у тех, кто впервые сталкивается с вычислениями на микрокомпьютерах. Я надеюсь, что как студент, так и просто заинтересованный читатель найдут эту книгу полезной для себя, прежде чем они решатся обратиться к более специальным работам по микрокомпьютерам. Но даже и тогда книга останется для них полезной как источник быстрой справки.

В книге рассматривается работа 8-разрядных микропроцессоров, создаваемых на базе новейшей технологии, основные принципы функционирования которых применимы к микропроцессорам с любой длиной слова. Для микрокомпьютера не требуется использование специального набора команд. Наоборот, я пытался показать, как Вы сможете воспользоваться любым набором реальных команд для работы с ним. Думаю, что даже пользователю, работающему со специальными языками высокого уровня, будет небезынтересно узнать, что происходит внутри его микрокомпьютера.

А. Вуд

---

* К настоящему моменту это заявление автора уже устарело. Микросхемы в виде так называемых заказных больших и сверхбольших интегральных схем благодаря успехам в автоматизации их проектирования остаются вполне конкурентоспособными с микропроцессорами, а по некоторым показателям и превосходят их. *(Прим. пер.)*

ГЛАВА ПЕРВАЯ

Кремниевые микросхемы

Что такое микропроцессоры и почему они так важны?

Микропроцессор, создаваемый как единый электронный компонент, является сердцем вычислительной машины. Именно микропроцессоры позволили микро-ЭВМ шагнуть туда, где никогда прежде не бывала вычислительная машина, — выйти в реальный мир.

Микропроцессор — это часть среды, которой он управляет, вместо того, чтобы быть отделенным от нее подобно первоначальному центральному блоку вычислительной машины или позже микро-ЭВМ. Микропроцессоры, ориентированные на решение отдельных, вполне определенных задач (те, что выполняют только одну работу, на которую они настроены, или, как говорят, запрограммированы изготовителями), уже появились в домашних телевизионных играх, игрушках, калькуляторах телефонных расходов и в других бытовых изделиях. Вскоре их можно будет встретить в большинстве домов внутри стиральных машин, в телевизионных приемниках, стереофонических системах, системах центрального отопления, системах сигнализации — фактически везде, где их применение приведет к оптимизации функционирования системы или к экономии используемых компонентов.

Телевизионные игры последнего времени могут «перепрограммироваться» с помощью инструкций для разных игр, содержащихся в сменных «модулях памяти». Некоторые из них могут даже программироваться непосредственно пользователем.

Наверное наиболее заманчивый вариант современного использования микропроцессоров, программируемых пользователем, — это их применение в персональной микро-ЭВМ. В настоящее время каждый желающий может приобрести в личное пользование микро-ЭВМ, которая компактна и недорога. Микропроцессор — это сердце микро-ЭВМ, он играет роль центрального обрабатывающего устройства, или центрального процессора (ЦП).

Это стало реальностью благодаря последним достижениям в полупроводниковой технологии, позволившим размещать электронные схемы огромной сложности на очень маленькой площади с потреблением совсем небольшой мощности и с низкими затратами на изготовление.

Схемы такого типа, называемые полупроводниковыми микросхемами, впервые стали применяться в карман-

Рис. 1.1. Отдельные полупроводниковые приборы (*а*) и интегральная схема с двухрядным расположением выводов (*б*):

1 — собственно интегральная схема; *2* — пластмассовый или керамический корпус, защищающий схему, с 28 выводами

ных калькуляторах, затем в электронных часах с цифровым отсчетом и, наконец, в микро-ЭВМ.

Микро-ЭВМ состоит из микросхемы центрального процессора и нескольких микросхем для таких функций, как память и управление вводом и выводом.

Первые полупроводниковые приборы выпускались заключенными в самостоятельные корпуса в виде отдельных изделий: транзисторов, диодов и т. п. (рис. 1.1, *а*). В 1958 г. были изобретены так называемые интегральные микросхемы (ИМС). Они представляли собой законченные электронные схемы, содержащие транзисторы и иногда диоды, резисторы и конденсаторы, размещенные целиком на одном кристалле из кремния, занимая, как правило, площадь около 65 мм^2.

Преимущества ИМС перед отдельными приборами и компонентами схем состоят в их небольшом размере, низкой стоимости и высокой надежности. Без такой миниатюризации персональная ЭВМ не была бы реальной не только потому, что благодаря ей физические размеры сокращены до обозримых масштабов и сэкономлены средства на изготовление корпусов, проводку и занимаемые площади, но и потому, что миниатюрные схемы потребляют меньшую мощность и позволяют ускорить процесс вычислений, поскольку электрическим сигналам не

приходится преодолевать излишне большие расстояния!. Кроме того, облегчается управление различными устройствами, входящими в состав ЭВМ.

Почему микропроцессы такие недорогие?

Снижение стоимости микропроцессоров вследствие применения ИМС происходит не только из-за их малых размеров. Большая доля экономии заложена в производстве самих микросхем. После разработки микросхемы и определения изготавливающего оборудования уже нет большого различия в стоимости производства одной микросхемы, содержащей 10 активных компонентов или 10 000 при условии, что они располагаются на кремниевом кристалле одного и того же размера. При изготовлении такого кристалла требуется практически одно и то же число технологических этапов в каждом случае, поэтому производственные расходы примерно одинаковы[1]. Это означает, что сердце ЭВМ, микропроцессор, можно сейчас купить по цене, примерно равной первоначальной стоимости отдельного транзистора.

Сколько компонентов можно разместить на одном кристалле?

Технология изготовления первых ИМС допускала размещение всего нескольких (максимум 10) активных элементов на одном кристалле. Такие ИМС стали относить к схемам с малой степенью интеграции (SSI—Small Scale Integration). Позже добились размещения до 100 активных элементов — ИМС со средней степенью интеграции (MSI—Medium Scale Integration). Технология изготовления современных микросхем допускает размещение гораздо большего числа активных элементов на одном кристалле. Микросхемы с числом элементов, большим 100, относят к ИМС с большой степенью интеграции (LSI—Large Scale Integratioh)[2].

---

[1] Имеется в виду, что в производственные расходы не входит стоимость разработки самой электрической схемы, которая, естественно, не одинакова в двух рассматриваемых автором случаях. Однако появившиеся в последние годы автоматизированные системы проектирования микросхем позволили резко снизить стоимость и этого этапа разработки. *(Прим. пер.)*

[2] Или к большим интегральным схемам (БИС).

На кристалле площадью в четверть квадратного дюйма уже удалось разместить 30 тыс. активных элементов, и ожидается, что это число дойдет до 1 млн. к 1985 г. Термин сверхбольшая степень интеграции (VLSI—Very Large Scale Integration) иногда применяется для обозначения микросхем с тысячами активных элементов[1]. Прогресс, происшедший в этой области, можно легко оценить, если представить себе, что в 50-е годы типичная ЭВМ содержала около 4000 электронных ламп, и их приходилось заменять примерно по 40 шт. в неделю. Ранее считалось, что стоимость ЭВМ пропорциональна квадрату ее потребляемой мощности. С появлением микро-ЭВМ это правило нарушилось. Персональные ЭВМ стоимостью менее 1 тыс. фунтов стерлингов в настоящее время обладают большей производительностью по сравнению с производительностью ЭВМ стоимостью в несколько сотен тысяч фунтов стерлингов, выпускавшихся несколько лет назад.

Что именно подразумевается под интегральной микросхемой с большой степенью интеграции?

Интегральная микросхема с большой степенью интеграции — это полностью функционально законченная электронная схема, изготовленная в одном корпусе. Существуют два основных вида интегральных микросхем. К первому из них относятся аналоговые микросхемы, применяемые, например, в качестве усилителей в системах звуковоспроизведения и в других линейных системах, а ко второму — переключающие, или дискретные, применяющиеся в микропроцессорах и в других дискретных устройствах. Чтобы разобраться в работе микропроцессора, необходимо изучить лишь дискретные схемы. Компонентами интегральных микросхем такого типа являются: ключевые элементы (транзисторы) и иногда элементы, ограничивающие протекание электрического тока (резисторы), элементы для сохранения зарядов для последующего использования (конденсаторы) и элементы, допускающие продвижение зарядов (тока) лишь в одном направлении (диоды).

[1] Или сверхбольших интегральных схем (СБИС). Кристалл рекордной сложности (430 тыс. транзисторов на площади в 40 мм²) разработан по технологии «кремний-на-сапфире» специалистами фирмы Hewlett Packard в 1981 г. *(Прим. пер.)*

Что означают сокращения: ДТЛ, ТТЛ, *n*-МОП, *p*-МОП, КМОП?

Основными элементами первых дискретных ИС являлись диоды и транзисторы; этот тип логики известен как диодно-транзисторная логика (ДТЛ). Позднее в схемах устройств диоды были заменены транзисторами, и такие устройства стали относить к схемам транзисторно-транзисторной логики (ТТЛ). Производство отдельных БИС для микропроцессоров оказалось возможным благодаря чрезвычайной миниатюризации, пришедшей вместе с технологией производства полевых транзисторов (FETs — field-effect transistors) особого типа, известных как МОП-транзисторы, или транзисторы, изготовленные по МОП-технологии (металл — окисел — полупроводник). Их работа отличается от работы обычных (биполярных) транзисторов, используемых в схемах ТТЛ, и более доступна пониманию, чем работа биполярных транзисторов. Полевые (униполярные, или канальные) МОП-транзисторы могут применяться в качестве усилителей, так же как и биполярные. Однако в микропроцессорах нас интересуют только их переключательные свойства.

Существуют два основных типа МОП-транзисторов. Первый тип представляет собой ключ, который нормально (при отсутствии входного сигнала) находится во включенном положении (нормально проводящий транзистор) и выключается только при подаче на вход (затвор) управляющего сигнала (потенциала). Нормально проводящие (нормально открытые) МОП-транзисторы называются транзисторами с обеднением канала.

Другой тип проще в изготовлении с применением технологии производства БИС, широко используемых в микро-ЭВМ. Это МОП-транзисторы с индуцированным каналом, или с обогащением канала. Они в нормальном состоянии (при нулевом смещении на затворе) выключены (нормально непроводящие, или нормально закрытые, транзисторы) и включаются только при подаче потенциала на вход (затвор) транзистора.

Как нормально открытые, так и нормально закрытые МОП-транзисторы могут быть *p*-канальными (*p*-МОП-транзисторы) или *n*-канальными (*n*-МОП-транзисторы) в зависимости от того, какой тип канала является проводящим[1]. На рис. 1.2 приведена схема *p*-канального нормально закрытого МОП-транзистора (с обогащением канала).

Заряды протекают через *p*-канальный МОП-транзистор только в том случае, если на затвор подается отрицательный потенциал по отношению к подложке (основание кремниевого кристалла, на котором построен транзистор), соединенной в *p*-канальном МОП-транзисторе с положительным полюсом источника питания U_+ (рис. 1.2). Следовательно, *p*-канальный МОП-транзистор включен, если затвор соединен с землей (нулевой потенциал), и выключен, если затвор соединен с положительным полюсом источника питания U_+ или изолирован (рис. 1.3).

На рис. 1.4 показан *n*-канальный нормально закрытый МОП-транзистор (с обогащением канала). Он содержит две области *n*-ти-

[1] Канал — тонкий поверхностный слой между областями истока и стока. *(Прим. пер.)*

па, полученные путем диффузии в подложку *p*-типа. Электроны не могут создавать проводимость от истока к стоку до тех пор, пока затвор не получит положительный потенциал по отношению к подложке. Если затвор имеет отрицательный потенциал по отношению к подложке, транзистор закрыт (рис. 1.5).

Заметим, что в обоих типах канального МОП-транзистора ток не протекает постоянно в затвор или из затвора. Именно напряже-

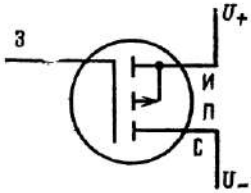


Рис. 1.2. Схема *p*-канального МОП-транзистора:

З — затвор; *И* — исток; *П* — подложка; *С* — сток (исток подключен к положительному полюсу источника питания, сток — к отрицательному)

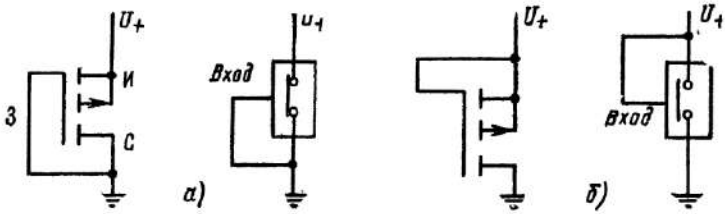


Рис. 1.3. *p*-канальный МОП-транзистор в качестве ключа:
а — ключ включен (транзистор открыт); *б* — ключ выключен (транзистор заперт)

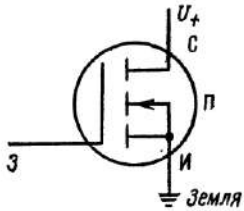


Рис. 1.4. Схема *n*-канального МОП-транзистора

ние, или электрическое поле, заставляет транзистор переходить в открытое или запертое состояние. Отсюда название «полевой» транзистор. Для более широкого ознакомления с технологией изготовления ИМС и с механизмами работы транзисторов можно обратиться к следующим изданиям: «Вопросы и ответы по интегральным микросхемам» (Questions and Answers on Integrated Circuits) и «Вопросы и ответы по транзисторам» (Questions and Answers on Transistors), опубликованным в серии технических новинок (Newness Technical Books) [1].

[1] См. также дополнительную литературу в конце книги. *(Прим. пер.)*

Оба типа МОП-транзисторов могут использоваться при построении логических схем в качестве ключей, близких к идеальным. Когда они включены, их сопротивление практически равно нулю, а когда выключены — оно составляет около миллиона миллионов ом (теоретически бесконечно велико). Однако МОП-транзисторы можно также использовать в качестве резисторов. Если канал транзистора сделать более длинным и узким, чем обычный, и его затвор соединить таким

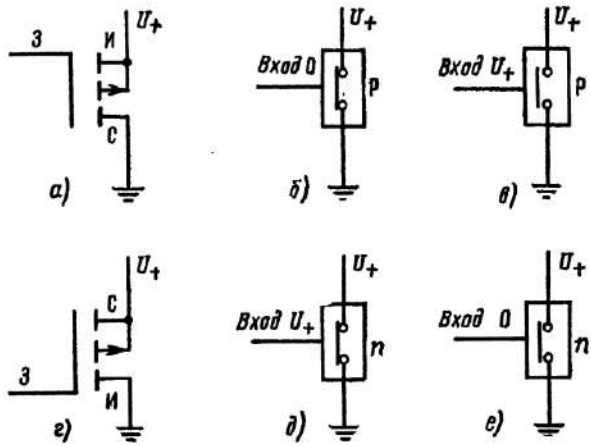


Рис. 1.5. Нормально закрытые МОП-транзисторы в качестве ключей:

а — обозначение *р*-канального транзистора; *б* — транзистор проводит, когда затвор соединен с землей (нулевой потенциал); *в* — транзистор не проводит, если затвор соединен с положительным полюсом источника питания U_+; *г* — обозначение *n*-канального транзистора; *д* — транзистор проводит, когда затвор соединен с положительным полюсом источника питания U_+; *е* — транзистор не проводит, когда затвор соединен с землей (нулевой потенциал)

образом, чтобы поддерживать полевой МОП-транзистор во включенном состоянии, то ток через транзистор будет проходить, но цепь транзистора будет обладать сопротивлением.

Затвор полевого МОП-транзистора можно использовать также и для накопления зарядов, поскольку между затвором и каналом ток полностью отсутствует благодаря очень хорошим электроизолирующим свойствам слоя окиси кремния между ними.

Каким образом используются МОП-транзисторы в интегральных микросхемах?

Интегральные микросхемы строятся путем реализации всех компонентов и связывающих их звеньев одновременно на одном кристалле (подложке из кремния).

Площадь кристалла обычно не превышает 150 мм². С таким устройством было бы трудно работать и его легко можно было бы повредить, если бы оно не было заключено в пластмассовый или керамический корпус с выводами, расположенными в два ряда (по одному ряду с каждой стороны корпуса, см. рис. 1.1, *б*). Такая организация выводов называется двухрядной (DIL — Dual-In-Line), а упаковка в корпусе с двухрядным расположением выводов называется двухрядной упаковкой (DIP — Dual — In-Line-Package). Обычно БИС имеют 8, 14, 16, 24, 28 или 40 выводов, расположенных с двух сторон корпуса по 4, 7, 8, 12, 14 или 20 с каждой стороны соответственно. Большинство микропроцессоров проектируется в корпусах с 40 выводами с двухрядным их расположением.

Схемы некоторых БИС построены из так называемых дополняющих пар канальных МОП-транзисторов *p*- и *n*-типов. Они носят название дополняющих, или комплементарных, МОП-структур (КМОП-структур). Можно приобрести БИС, в которых используются только МОП-транзисторы *p*-типа, известные как *p*-МОП-БИС, или другие, в которых используются только МОП-транзисторы *n*-типа, известные как *n*-МОП-БИС.

БИС микропроцессоров обычно содержат КМОП-структуры и *n*-МОП-транзисторы. Можно также встретить и другие БИС, меньшего размера, используемые в микропроцессорной технике, причем некоторые из них могли бы быть с успехом построены на базе транзисторно-транзисторной логики (ТТЛ). В книге «Вопросы и ответы по интегральным микросхемам» подробно рассмотрены особенности различных видов технологии производства интегральных микросхем, в частности методы диффузии.

Почему МОП-технология открывает возможности бóльшей миниатюризации, чем технология производства ТТЛ?

МОП-транзистор занимает меньшую площадь на кристалле, чем биполярный транзистор, поскольку для размещения стока и истока МОП-транзистора требуется всего одна диффузия.

Для биполярного транзистора понадобятся три диффузии и соответственно большая площадь для его размещения, так как область каждой диффузии должна быть внутри предыдущей.

Большая миниатюризация достигается также благодаря тому, что схемы на МОП-транзисторах значительно проще схем ТТЛ.

Как упоминалось ранее, с помощью МОП-транзисторов можно легко организовать резисторы, для которых потребуется совсем небольшая дополнительная площадь. Диффузионные резисторы, используемые в БИС на биполярных транзисторах, занимают намного больше места.

Техника разработки интегральных микросхем сводится к построению и многократному использованию одних и тех же базовых подсхем. Именно этот принцип стал распространенным среди изготовителей интегральных микросхем. Если рассмотреть кремниевую микросхему под микроскопом, то можно увидеть постоянно повторяющиеся комбинации одних и тех же базовых подсхем или узлов (рис. 1.6).

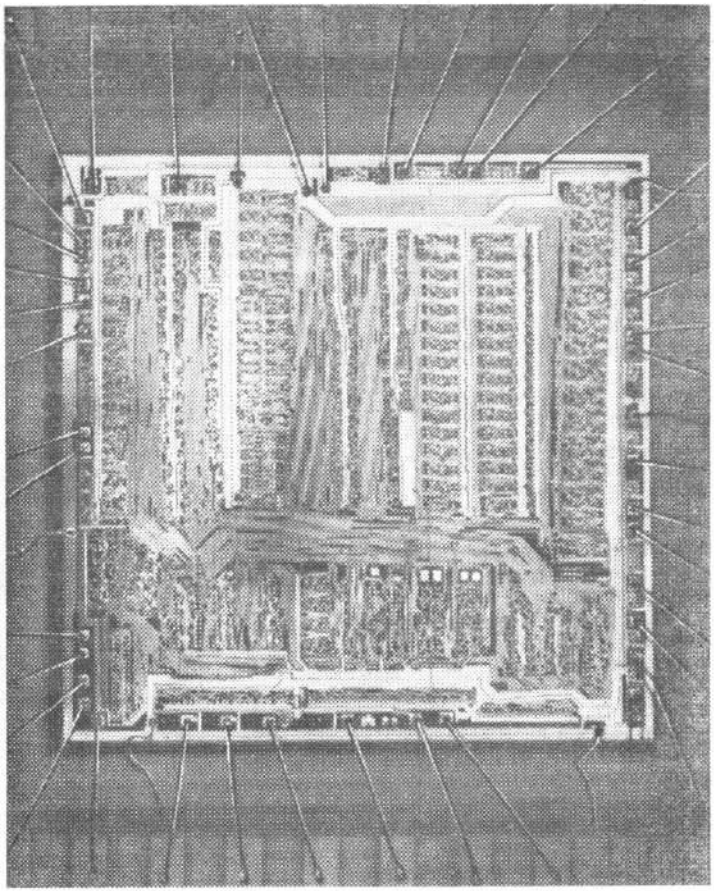


Рис. 1.6. Вид на микропроцессор со снятой защитной крышкой

Как производятся переключения?

Новичку в электронике легче всего начать изучение особенностей работы микро-ЭВМ с рассмотрения действия схем комплементарных МОП-структур (КМОП-структур), которые будут широко использоваться в дальнейшем при описании работы различных частей микро-ЭВМ. Основным типовым узлом (базовой подсхемой) КМОП-структур является инвертирующий ключ, или инвертор. Далее увидим, что путем соединения таких узлов различными способами и в различных сочетаниях можно построить любую часть микро-ЭВМ.

Прежде чем перейти к подробной схеме инвертора на основе КМОП-структур (КМОП-инвертора), рассмотрим упрощенную модель работы по переключению цепи, выполняемой инвертором. Для данных целей инвертор может рассматриваться как сблокированный в одном корпусе перключатель с двумя разрывами цепи, на выходе которого может быть сигнал ЛИБО U_+, ЛИБО «земля». Сблокированный означает, что обе части движутся вместе, как бы соединенные неким рычагом, причем если один контакт замкнут, то другой — разомкнут, и наоборот (рис. 1.7).

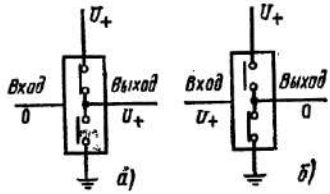


Рис. 1.7. Инвертор как переключатель:

а — вход соединен с землей; *б* — вход соединен с положительным полюсом источника питания

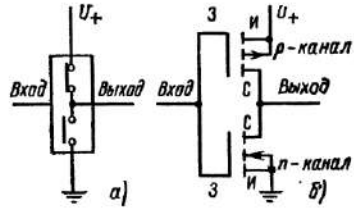


Рис. 1.8. КМОП-инвертор:

а — механическая модель; *б* — схема КМОП-структуры из двух транзисторов (дополняющая пара: верхний — *р*-канальный, нижний — *п*-канальный)

Если вход инвертора соединен с землей (нуль), переключатель соединяет выход с U_+ (рис. 1.7, *а*). Если вход соединен с U_+, выход переключателя соединен с землей (рис. 1.7, *б*).

Если бы это был механический переключатель, можно было бы предположить, что он приводится в действие электромагнитом и возвращается в прежнее положение какой-то специальной пружиной. Из схемы видно, что состояние выхода всегда противоположно состоянию входа. Это и называется инвертированием сигнала.

Однако в микро-ЭВМ не применяются ни электромагниты, ни механические переключатели; для осуществления переключений в них используются транзисторы. Типовая схема КМОП-инвертора вместе с соответствующим механическим аналогом приведена на рис. 1.8.

В схеме следует различать транзисторы *р*- и *п*-типов, действующие как переключатели. Затворы этих транзисторов соединены между собой, следовательно, когда один транзистор включен, другой должен быть выключен. Это так, поскольку, когда на вход схемы поступает сигнал U_+, оба затвора получают этот сигнал, который включает *п*-канальный МОП-транзистор и выключает *р*-канальный МОП-транзистор, в результате чего выход схемы соединяется с землей (получает нулевой потенциал) через *п*-канальный транзистор.

Когда оба затвора соединены с землей (получают нулевой потенциал), *п*-канальный транзистор выключается, а *р*-канальный включается. Таким образом, выход схемы соединяется с U_+ через *р*-канальный транзистор.

На рис. 1.9 приведены для сравнения схемы инверторов на базе ТТЛ, *р*-МОП-транзисторов и *п*-МОП-транзисторов. В работе *п*-МОП- и *р*-МОП-инверторов можно разобраться, сравнивая их с КМОП-ин-

вертором. Попытаемся предельно упростить n-МОП- и p-МОП-инверторы до простейших переключающих схем и посмотрим, как они работают.

Схему p-МОП-инвертора можно рассматривать как ключ между входным полюсом U_+ и выходом, связанным с землей через резистор. Когда на вход схемы поступает сигнал 0, ключ замыкает цепь, и на

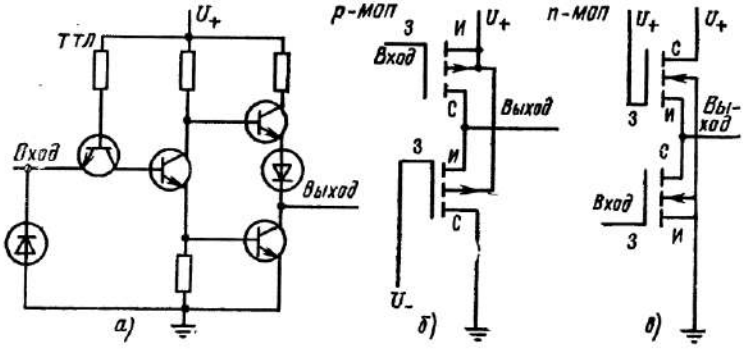


Рис. 1.9. Схемы инверторов на базе: элементов ТТЛ (*а*); p-МОП-транзисторов (нижний транзистор, поддерживаемый во включенном состоянии, играет роль небольшого сопротивления) (*б*); n-МОП-транзисторов (верхний транзистор, поддерживаемый во включенном состоянии, играет роль небольшого сопротивления) (*в*)

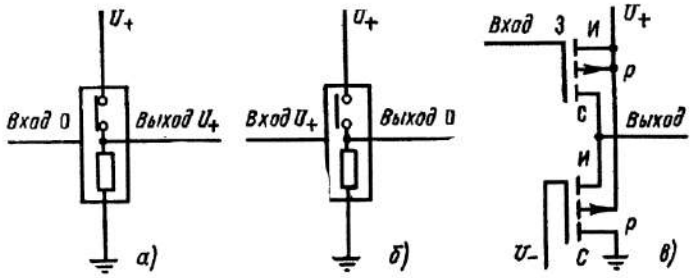


Рис. 1.10. Работа p-МОП-инвертора в качестве ключа

выходе появляется сигнал U_+ (рис. 1.10, *а*). Если на входе схемы имеется сигнал U_+, ключ открыт, и выход оказывается связанным с землей через резистор (рис. 1.10, *б*). Если в цепи выхода тока совсем нет или протекает очень маленький ток (например, когда выход соединен с затвором другого канального транзистора), на выходе схемы будет нулевой потенциал. На рис. 1.10, *в* приведена соответствующая схема, в которой верхний p-МОП-транзистор играет роль ключа, а нижний, поддерживаемый во включенном состоянии, играет роль нагрузки.

Схему n-МОП-инвертора на рис. 1.11 можно рассматривать как ключ между землей и выходом, связанным через резистор с положительным полюсом источника питания U_+. Когда на входе схемы имеется сигнал 0, ключ открыт, и выход схемы соединен с полюсом U_+ через резистор (рис. 1.11, *а*). Если в выходной цепи тока совсем нет или протекает очень небольшой ток, как в случае соединения выхода

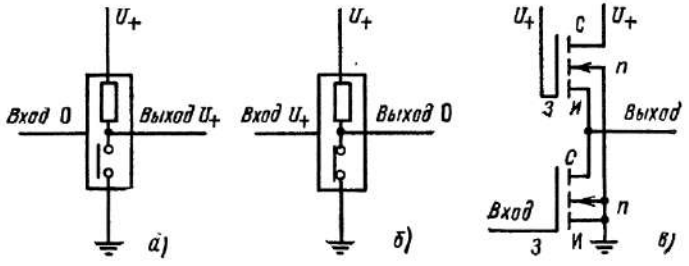


Рис. 1.11. Работа n-МОП-инвертора в качестве ключа

с затвором другого канального транзистора, то потенциал на выходе схемы будет равен U_+. Если вход схемы соединен с U_+, ключ закрыт и выход схемы соединен с землей.

На рис. 1.11, *в* приведена схема n-МОП-инвертора, в которой нижний транзистор играет роль ключа, коммутирующего выход при изменениях входного сигнала в соответствии со схемами на рис. 1.11, *а* и *б*, а верхний транзистор, поддерживаемый во включенном состоянии, играет роль нагрузки.

В схемах микро-ЭВМ используются переключатели любого из трех рассмотренных типов, поскольку каждому из них присущи характерные для него преимущества и недостатки. Наиболее распространенными в настоящее время являются n-МОП- и КМОП-схемы. Первые потребляют бóльшую мощность, чем вторые, из-за резисторов, потребляющих энергию даже в том случае, когда на выходе схемы формируется нулевой сигнал. Вместе с тем КМОП-схемы потребляют существенно меньшую энергию, чем другие типы схем, поскольку сами переключатели не потребляют никакой энергии. Схемы на основе p-МОП-транзисторов наименее быстродействующие и поэтому используются реже.

После более подробного знакомства с различными устройствами микро-ЭВМ можно рассмотреть особенности их работы в терминах переключателей (инверторов), построенных на базе комплементарных МОП-структур, так что неискушенным в области электроники не потребуется знакомство с работой каких-либо других компонентов схем, кроме рассмотренных переключателей. При желании можно самостоятельно убедиться в эквивалентности соответствующих n-МОП и p-МОП-структур.

Аппаратное обеспечение и его терминология

Какие микропроцессоры встречаются чаще всего?

Существует множество различных микропроцессоров, выпускаемых разными изготовителями. Чаще всего встречаются следующие модели: 6502, 6800, Z80 и 8080, поскольку это наиболее распространенные микропроцессоры в персональных микро-ЭВМ.

Модель 6502—это микросхема с 40 выводами в корпусе с двухрядным расположением выводов, построенная на базе *n*-МОП-технологии; используется в таких микро-ЭВМ, как KIM, Superboard, Challenger, Pet, Acorn и Apple II.

Модель 6800 — также микросхема с 40 выводами на базе *n*-МОП-технологии в корпусе с двухрядным расположением выводов; используется в таких микро-ЭВМ, как модель 77-68 и Southwest Technical Products 6800 Computer System.

Модель Z80—также микросхема с 40 выводами на базе *n*-МОП-технологии в корпусе с двухрядным расположением выводов; используется в таких микро-ЭВМ, как Tandy TRS80, Exidy Sorcerer, Research Machine's 380Z, Tuscan и Nascom.

Модель 8080 — предшественник модели Z80, используется в ABC 80, Triton и во многих других микро-ЭВМ.

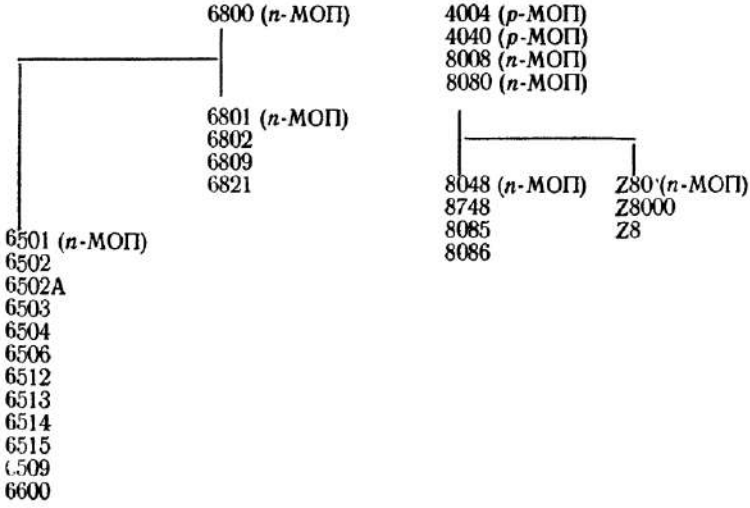
Модели Z80 и 6502 иногда относят к микропроцессорам третьего поколения, поскольку они принадлежат к семейству микропроцессоров, где каждый последующий член семейства является усовершенствованием предыдущего.

В идеальном случае хорошо было бы иметь один тип микропроцессора для всех видов применения, однако поскольку совершенствование микропроцессорной техники продолжается, такая возможность представляется пока весьма отдаленной.

Общепринятая аббревиатура для микропроцессорного устройства MPU (microprocessor unit) обозначает просто микропроцессор (МП). В микро-ЭВМ MPU используется в качестве центрального обрабатывающего устройства (CPU — central processing unit). Центральное обрабатывающее устройство координирует действия

и управляет работой отдельных частей микро-ЭВМ, выполняя также все арифметические и логические операции и осуществляя передачу информации от одного устройства к другому.

Существуют следующие два основных дерева семейств МП[1]:



Что еще требуется для микро-ЭВМ помимо микропроцессора?

Для микро-ЭВМ так же, как и для микропроцессора, требуется память, в которой хранится информация о том, как управлять процессом обработки информации с помощью команд программы пользователя. Этот тип памяти используется только для считывания информации и относится к памяти типа ROM (Read Only Memory),

[1] Изделия 8048, 8748 относят также к однокристальным 8-разрядным микро-ЭВМ (фирмы Intel). В состав первого входят: 8-разрядный ЦП, память команд емкостью 1К×8 слов, оперативное запоминающее устройство емкостью 64×8 слов (выполняющее функцию памяти данных), 27 линий ввода-вывода, 8-разрядный таймер-счетчик, генератор и формирователь тактовых импульсов, а также схемы сброса и прерываний. Кроме того, изделие 8048 имеет постоянное запоминающее устройство с масочным программированием, а изделие 8748 включает в свой состав программируемое ПЗУ пользователя с возможностью стирания информации. *(Прим. пер.)*

или к постоянной памяти. Постоянная память программируется при изготовлении, и ее содержимое не может меняться пользователем.

Кроме ROM, или постоянной памяти, для микро-ЭВМ требуется память, в которой пользователь может хранить данные и команды собственной программы и из которой ЭВМ считывает данные под действием управляющих команд, хранящихся в постоянной памяти. Эта память относится к типу памяти для считывания и записи[1], обозначаемой RAM. Первоначально она создавалась как память с произвольным доступом (RAM—Random Access Memory), однако постоянная память также является памятью с произвольным доступом, т. е. при обращении к ней нет необходимости в организации последовательного считывания информации, как при использовании кассеты с магнитной лентой (внешней памяти). Можно обращаться к любой ее части в любом порядке подобно перелистыванию страниц в справочнике.

Итак, будем считать ROM постоянной памятью, допускающей только считывание, или постоянным запоминающим устройством (ПЗУ), а RAM — оперативной памятью, т. е. памятью для считывания и записи, или оперативным запоминающим устройством (ОЗУ).

Что такое внешние устройства микро-ЭВМ и порты?

В микро-ЭВМ требуются специальные вводные и выводные устройства связи, или так называемые порты, для подсоединения ЦП к внешним устройствам, называемым также периферийным оборудованием. К числу последних относят: клавишный пульт, телевизионный монитор, печатающее устройство для выдачи обработанной информации на бумаге (копий на однократно используемом носителе). Порты[2], или устройства сопряжения ЦП с периферийным оборудованием, относят к интерфейсам;

---

[1] Или, иначе, к оперативной памяти (ОП). *(Прим. пер.)*

[2] Понятие «порт» (фонетическая калька с английского port) в настоящее время получило широкое распространение в переводной литературе по микроэлектронике. Это специальная схема (обычно часть микросхемы), обеспечивающая синхронный или асинхронный обмен данными. Например, изделие 8255 фирмы Intel (программируемый интерфейс внешних устройств) в одном из основных режимов работы можно рассматривать как три 8-разрядных порта. *(Прим. пер.)*

на практике прежде всего встречаются такие устройства, как ACIA, UART, интерфейс RS 232, кассетный интерфейс Канзас Сити (Kansas City) и контрольно-измерительный интерфейс IEEE-488.

Что такое шина?

Линии, по которым данные передаются к различным устройствам микро-ЭВМ, называются линиями передачи данных. Совокупность линий передачи данных называется шиной передачи данных. Обычно в микро-ЭВМ данные передаются к различным ее частям параллельно по восьми линиям; поэтому соответствующую шину называют 8-разрядной шиной передачи данных, так как по каждой линии может передаваться лишь один двоичный разряд (бит) информации.

Каковы функции синхрогенератора?

Чтобы все части (устройства) микро-ЭВМ действовали в правильной последовательности, необходим синхронизатор, или специальный генератор тактовых импульсов, синхронизирующий работу всей системы. Иногда микросхема ЦП содержит в себе и схему синхрогенератора, для работы которого необходимо подключение лишь нескольких внешних компонентов[1]. Однако некоторые микропроцессоры нуждаются в полной внешней схеме синхрогенератора.

Каким образом посылаются сигналы к различным устройствам ЭВМ?

Для передачи команд к различным устройствам микро-ЭВМ необходимы еще управляющие и адресные линии. Эти линии, как правило, отделены от линий передачи данных. Обычно используются 16 адресных линий, с помощью которых осуществляется возможность выбрать любой участок памяти, т. е. произвести ее адресацию. Совокупность этих 16 адресных линий составляет адресную шину.

Целиком вся система линий передачи данных, адресации, управления и питания называется шинной струк-

[1] В частности, кварцевого резонатора. *(Прим. пер.)*

турой. Были предприняты две основательные попытки разработать стандартную схему расположения таких линий для облегчения решения задачи компоновки и совместного подсоединения оборудования (устройств) различных изготовителей. Это шинная система S100 и система E78, или европейская шинная система.

Что подразумевается под аппаратным и программным обеспечением?

Аппаратное обеспечение, или аппаратура (hardware), — название, закрепленное за механическими и электронными устройствами микро-ЭВМ. Микропроцессор, память, клавиатура, дисплей для отображения визуальной информации (экран телевизора) — все это и есть аппаратное обеспечение, или аппаратура.

К программному (или математическому) обеспечению (software) относятся собственно программы. Меняя программу в микро-ЭВМ, можно заставить аппаратуру выполнять различные функции. Например, с помощью одной программы микро-ЭВМ можно использовать для телевизионной игры, с помощью другой—в качестве калькулятора, с помощью третьей — для регистрации ценных бумаг или оформления счетов, с помощью четвертой — для охранной сигнализации, предупреждающей о появлении грабителей, и т. д.

Какие типы запоминающих устройств используются в микро-ЭВМ?

Различные программы работы микро-ЭВМ обычно хранятся во вспомогательных запоминающих устройствах, относящихся к внешним запоминающим устройствам, таким как устройство записи на магнитную ленту. Эти программы поступают в микро-ЭВМ по мере необходимости, поскольку, за исключением некоторых сугубо специальных случаев управления ее работой, совершенно очевидно, что внутренней памяти микро-ЭВМ не хватит для хранения и выполнения всех программ, которые вы захотите использовать. А если бы даже и хватило, ЭВМ пришлось бы быть постоянно включенной, так как ОЗУ теряет свою информацию при выключении питающего напряжения, и при повторном включении его снова нужно программировать (память, не сохраняющая информацию при выключении).

Постоянное запоминающее устройство не теряет хранящуюся в нем информацию. Однако все программы в нем хранить нецелесообразно, так как в этом случае пользователь зависел бы от возможностей изготовителя предоставить ему те или иные программы. Чтобы помочь пользователю редактировать или прогонять и другие программы наряду с теми, которыми он пользуется наиболее часто, существует так называемая постоянная память, программируемая пользователем (PROM—Programmable Read Only Memory), или программируемое ПЗУ (ППЗУ). Особенностью памяти, программируемой пользователем, является то, что программируют ее 1 раз в соответствии с собственными нуждами и потом используют как постоянную память. Это довольно дорогое устройство. Его иногда называют еще электрически программируемым постоянным запоминающим устройством (ЭППЗУ), но не нужно путать его с устройством памяти типа EPROM (Erasable Programmable Read Only Memory), или с перепрограммируемым ПЗУ, под которым подразумевается программируемая постоянная память с возможностью стирания. Ее можно запрограммировать в основном так же, как и просто программируемое ПЗУ, и пользоваться затем как постоянным запоминающим устройством. Однако, если в дальнейшем потребуется изменить содержимое этой памяти, для этой цели в верхней части устройства есть кварцевое окошко, сквозь которое, если его подвергнуть ультрафиолетовому облучению определенной интенсивности и длины волны, содержимое памяти сотрется, и его снова можно будет использовать для дальнейшего программирования. Иногда такую память обозначают UVEPROM.

Итак, подведем некоторые итоги. Программное обеспечение, используемое при работе с каждой программой (например, управляющие программы), лучше всего хранить в ПЗУ того или иного типа. Такой вид обеспечения иногда называют программно-аппаратным обеспечением (т. е. програмным обеспечением, жестко зафиксированным в аппаратуре). Все необходимые рабочие программы должны переписываться в оперативную память всякий раз, когда они требуются для выполнения на ЭВМ. Такие готовые для непосредственного применения программы могут храниться различными способами. Они могут быть записаны на бумаге и вводиться в микроЭВМ с помощью специальных кнопок в простой системе

или с помощью клавиатуры (keyboard) в более совершенной системе. Они могут храниться также в записи на магнитной ленте (во внешнем запоминающем устройстве) и переписываться в микро-ЭВМ через входной порт последовательного ввода, называемый адаптером последовательного интерфейса асинхронного обмена (ACIA—Asynchronous Communication Interface Adapter)[1]. Иногда для этой цели используется универсальный асинхронный приемопередатчик (УАПП) или UAPT (Universal Asynchronous Receiver Transmitter)[2].

Магнитофонный интерфейс иногда определяется как соответствующий определенному стандарту Канзас Сити, или CUTS (Computer Users' Tape System). Этот стандарт оговоривает особый способ хранения данных на ленточном носителе в специальной системе CUTS ввода данных с магнитной ленты в микро-ЭВМ, определяющий последовательности модуляций для представления данных в двоичной форме в виде последовательностей единиц и нулей.

Для выполнения микро-ЭВМ операций над данными, записанными на магнитной ленте, требуются специальные порты ввода, поскольку, хоть микро-ЭВМ и может обрабатывать данные параллельно с каждой из восьми ее дорожек сразу, информация поступает с ленты только поразрядно последовательно по мере прохождения лентой считывающей головки. Хотя на обычных устройствах записи на магнитную ленту может быть достигнута скорость около 300 двоичных разрядов в секунду (300 бод), а на хороших накопителях — до 1200 бод, все-таки может потребоваться некоторое время, чтобы ввести программу с кассеты, особенно если для этого требуется пропустить ее всю целиком. (Заметим: 1 бод равен одному разряду в секунду; при этом управляющие сигналы не учитываются, а учитываются только данные, необходимые для обработки адреса в памяти и команды).

Более оперативный, хотя и более дорогостоящий метод — это хранение информации на гибком диске, или на флоппи-диске (floppy disk[3], или на мини-флоппи. Для

---

[1] Описание работы ACIA приведено в [1]. *(Прим. пер.)*

[2] Выпускаемый в виде отдельной микросхемы [2]. *(Прим. пер.)*

[3] Гибкий магнитный диск—это пластинка из синтетического материала с пригодным для магнитной записи сигнала оксидным слоем. *(Прим. пер.)*

этого вида магнитной записи требуется большая память, специальный порт последовательного ввода-вывода и достаточно сложная система управления[1]. Хотя информация записывается и считывается в последовательной форме, диск обеспечивает почти немедленный доступ к любой зоне, существенно более быстрый по сравнению с доступом при записи на магнитную ленту.

С каким еще оборудованием придется встретиться?

В качестве устройства ввода информации в микро-ЭВМ используется либо набор кнопок или ключей (в самой простой машине), либо тот или иной вид клавиатуры или клавишного пульта. Из них простейшей является шестнадцатеричная клавишная панель. В такой панели имеется 16 клавишей для ввода команд программы, данных для обработки и адресов (рис. 2.1). Кроме того, в ней также должно быть несколько управляющих ключей или кнопок для запуска и прогона программ.

Более совершенным устройством ввода является полная клавиатура пишущей машинки или клавишный пульт QWERTY (первые шесть букв слева верхнего ряда клавишей).

При нажатии любой из клавишей пульта соответствующий сигнал преобразуется в код для передачи по шине данных. В качестве такого кода обычно используется американский стандартный код для обмена информацией ASCII (American Standard Code for Information Interchange)[2].

Каждый тип пульта ввода данных сопрягается (соединяется) с шиной через порт ввода, преобразующий данные в параллельную форму. Подобные устройства сопряжения (интерфейсы) известны как адаптеры внешнего интерфейса[3] PIA (Peripheral Interface Adapters), или устройства (порты) параллельного ввода-вывода PIO (Parallel Input/Output). Они выполняют простую функцию, аналогичную той, которая реализуется упоми-

---

[1] Некоторые данные, в том числе схема сопряжения микро-ЭВМ с флоппи-диском, приведены в [3]. *(Прим. пер.)*

[2] Шестнадцатеричное представление символов кода ASCII приведено в [2]. *(Прим. пер.)*

[3] Описание и характеристики адаптера внешнего интерфейса PIA приведены в [1]. *(Прим. пер.)*

навшимися ранее портами последовательного ввода. Однако информация в данном случае передается в параллельной форме (т. е. более одного двоичного разряда в единицу времени), тогда как устройства вывода на магнитную ленту, телетайп, дисплей или устройство визуального отображения VDU (Visual Displau Unit), такие как адаптер последовательного интерфейса асинхронного обмена ACIA и универсальный асинхронный приемопередатчик UART, преобразуют информацию в последовательную форму. Кажется, что информация на экране телевизора видна вся одновременно, но это иллюзия. Изображение строится лучом, движущимся от одной позиции к следующей вдоль каждой строки изображения по очереди, т. е. информация на экране дисплея воспроизводится в последовательной форме.

C	D	E	F
8	9	A	B
4	5	6	7
0	1	2	3

Рис. 2.1. Типовая шестнадцатеричная клавишная панель

Линии передачи данных обычно буферируются[1], прежде чем они будут подключены к соответствующим портам. Буфер — это специальное устройство, обеспечивающее электрическое согласование частей системы. Буферы с тремя состояниями выхода (тристабильные буферы) в одном из режимов работы могут обеспечить электрическую изоляцию (практически отрыв) одной части системы от другой.

Простейшие из существующих микро-ЭВМ не имеют портов вывода; их линии передачи данных подключены к светоизлучающим диодам (LED — light-emitting diodes) через соответствующие буферы.

Некоторые устройства ввода и вывода снабжаются специальными схемами, известными как средства хэнд-

[1] Под буферированием в данном случае подразумевается обеспечение схемотехническими средствами выравнивания нагрузки для каждой линии шин адреса или данных и недопущение ее превышения в любой момент времени. Для этих целей обычно используются специальные схемы либо специальные устройства, называемые шинными формирователями. Описание работы и схемы соответствующих устройств приведены в [4]. *(Прим. пер.)*

шейка (handshake facilities). Хэндшейк буквально означает «рукопожатие». Под термином «хэндшейкинг» (handshaking) подразумевают специальную процедуру управления обменом, при которой внешние устройства и центральный микропроцессорный элемент обладают возможностью «спросить» друг друга, готовы ли они к обмену информацией, и данные передаются только при ответе «да».

Если необходимо, чтобы микро-ЭВМ обладала способностью обмениваться информацией с расположенной на некотором расстоянии другой микро-ЭВМ, можно воспользоваться телефонной связью при наличии модулятора/демодулятора, называемого МОДЕМ (MODEM), который преобразует информацию с адаптера внешнего интерфейса микро-ЭВМ в звуковые сигналы, посылаемые в телефонную линию через акустическое соединительное устройство.

Какова типовая структура аппаратного оборудования микро-ЭВМ?

Объем аппаратуры зависит от того, насколько простой или сложной задумана соответствующая микро-ЭВМ. Пример простейшей вычислительной системы на базе микропроцессора приведен на рис. 2.2.

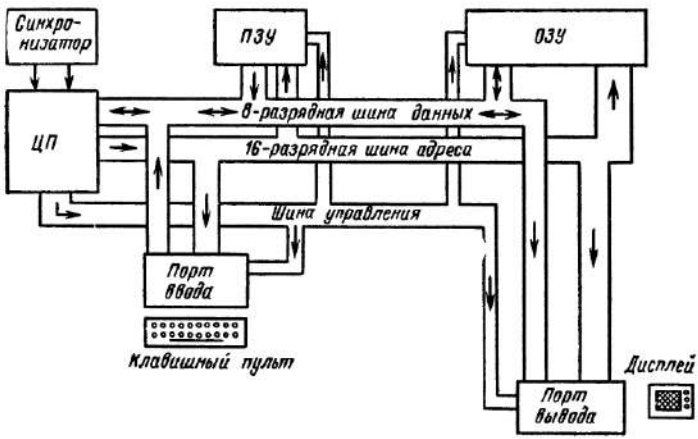


Рис. 2.2. Структура простой микро-ЭВМ

Программное обеспечение и его терминология

Что такое программное обеспечение?

В письменном виде или в изображении на экране дисплея — это адреса (или порядковые номера), команды и данные в виде слов, кодов или чисел в зависимости от типа конкретного используемого языка программирования.

Например, многие микро-ЭВМ ориентированы на использование языка программирования БЕЙСИК. Это язык высокого уровня. Программа сложения двух чисел на языке БЕЙСИК имеет следующий вид:

Порядковые номера	Команды и данные		
01	LET	A = 7	(ПРИСВОИТЬ)
02	LET	B = 11	(ПРИСВОИТЬ)
03	LET	C = A + B	(ПРИСВОИТЬ)
04	PRINT	C	(ПЕЧАТАТЬ)
05	END		(КОНЕЦ)

В таком виде информация не может восприниматься микропроцессором. Поэтому специальная программа, имеющаяся в ПЗУ, называемая интерпретатором, или компилятором[1], преобразует программу на языке высокого уровня в программу на языке машины для обеспечения ее прохождения (выполнения).

Если отказаться от языков высокого уровня, можно использовать символический машинно-ориентированный язык, или так называемый язык ассемблера (assembly language). Он может быть применен для микро-ЭВМ с меньшим объемом памяти, но тогда придется выполнить дополнительную работу при написании программы для такой машины. Приведенная выше программа сложе-

[1] Общее название программы перевода исходной программы на языке высокого уровня в объектную программу на машинном языке — транслятор. Различают трансляторы интерпретирующего типа, когда процесс трансляции совмещается с выполнением составленной им объектной программы, и компилирующего типа, когда в процессе трансляции получают объектную программу, которая затем может выполняться по мере необходимости. *(Прим. пер.)*

ния двух чисел на языке ассемблера[1] выглядит следующим образом:

Мнемонические символы команд	Адреса в памяти
LDA	ячейка 102
ADC	ячейка 203
STA	ячейка 101
BRK	

где LDA означает загрузить (load), ADC — сложить (add), STA — записать в память (store), BRK — прекратить работу (stop).

Специальная программа, называемая программой ассемблера, или просто ассемблером, преобразует символические выражения на языке ассемблера в программу на языке машины для обеспечения ее прохождения (выполнения).

Следующий шаг на пути приближения к языку машины состоит в программировании в шестнадцатеричном коде, к которому приходится прибегать в недорогих микро-ЭВМ. Рассмотренная программа сложения двух чисел в шестнадцатеричном коде выглядит следующим образом:

Шестнадцатеричный код операции	Адрес в памяти, на который имеется указание в команде		
AD	11	00	Действительные
6D	CB	00	номера должны быть
8D	65	00	снова занесены
00	—	—	в память

Специальная управляющая программа, называемая монитором и хранимая в постоянной памяти, переводит эти выражения после введения программы в выражения на языке машины.

Если Вы имеете дело с простейшей микро-ЭВМ с ключами (кнопками) в качестве входов, то программы возможно придется вводить непосредственно на языке машины с помощью ключей ввода. Эта же программа сло-

[1] Символические машинно-ориентированные языки относят к языкам низкого уровня и называют также автокодами. Из множества автокодов принято выделять автокоды типа 1 : 1. К этой группе относятся такие автокоды, каждое предложение которых порождает не более одного машинного слова. При составлении программы на языках этого типа не требуется указаний о соответствии между символическими и истинными адресами. К другой группе автокодов относятся автокоды с адресными выражениями. *(Прим. ред.)*

жения двух чисел в машинных кодах (на языке машины) выглядит следующим образом:

Двоичный код операции		Адрес в памяти, на который имеется указание в команде			
1010	1101	0110	0110	0000	0000
0011	1100	1100	1011	0000	0000
1000	1101	0110	0101	0000	0000
0000	0000	—	—	—	—

Каждая строчка, т. е. код операции и адрес, на который имеется ссылка (если он имеется), называется командой.

Коды операций, используемые в данном примере, — это коды микропроцессора модели 6502. Его команда сложения имеет смысл «сложить с учетом переноса» и требует, чтобы отсутствовали переносы от предыдущих вычислений (флаги переносов были чистыми), прежде чем команда сложения начнет выполняться [смысл понятия «флаг переноса» будет уточнен позже (см. гл. 5)]. Для простоты в приведенном выше примере предыдущие переносы не рассматривались, но в дальнейшем о них будет сказано подробнее. В некоторых микропроцессорах имеется, кроме того, отдельная команда СЛОЖИТЬ БЕЗ ПЕРЕНОСА, и в этом случае флаг переноса не нужно очищать перед ее применением. В примерах, приведенных в этой главе, будем пользоваться командой СЛОЖИТЬ С УЧЕТОМ ПЕРЕНОСА, а в следующей главе покажем, как можно пользоваться этими двумя типами команд, если они предусмотрены в микропроцессоре.

К сожалению, для различных микропроцессоров коды операций различны, и, если программирование осуществляется на языках низкого уровня (в шестнадцатеричном или машинном коде), необходимо знать коды операций системы команд для микропроцессора, которым пользуетесь.

Краткие выводы

Язык высокого уровня $(C=A+B)$ ⟶ программа компилятор ⟶ машинный язык ⟶ микропроцессор — в 5 раз быстрее, чем

Язык ассемблера (символический) ⟶ программа ассемблера ⟶ машинный язык ⟶ микропроцессор — в 10 раз быстрее, чем

Шестнадцатеричное программирование (FF и т. д.) → программа монитор → машинный язык → микропроцессор — в 4 раза быстрее, чем

Двоичный машинный язык (0010 и т. д.) → программа загрузчик → микропроцессор.

Нетрудно заметить, что программирование на языке высокого уровня дает почти 5-кратное повышение скорости составления программы по сравнению с программированием на языке ассемблера, или (5×10) 50-кратное повышение скорости по сравнению с программированием в шестнадцатеричном коде, или (5×10×4) 200-кратное повышение скорости по сравнению с программированием на языке машины.

Зачем нужны команды в машинном коде?

Как было показано ранее, аппаратное обеспечение по существу действительно очень простое, поскольку построено на базе множества ключей (переключателей), которые могут находиться только в двух электрических состояниях: ВКЛЮЧЕН и ВЫКЛЮЧЕН. Было бы очень сложно сконструировать надежную электрическую систему, способную четко идентифицировать любой из 10 уровней напряжения, используемых для представления десятичных чисел, количественный эквивалент которых выражается с помощью цифр от 0 до 9 в обычной десятичной системе счисления. Поэтому программное обеспечение призвано свести любую задачу к последовательностям выполнения операций ВКЛЮЧЕНО и ВЫКЛЮЧЕНО, требующим для соответствующего представления только двух цифр (обычно 0 для ВЫКЛЮЧЕНО и 1 для ВКЛЮЧЕНО).

Микропроцессор производит вычисления «неуклюжим», с точки зрения человека, способом, а именно, с использованием двоичной системы счисления. Основанием двоичной системы счисления является 2 вместо 10 в обычной системе. Это означает, что перенос в следующий (старший) разряд числа необходимо выполнять, когда значение в данном разряде достигнет 2, так же как в десятичной системе счисления единица переносится в следующий слева разряд каждый раз, когда значение разряда достигнет 10.

Так, начиная с нуля и увеличивая каждый раз значение количественного эквивалента двоичного числа на

единицу, можно получить следующую последовательность двоичных чисел:

Начало	0000
Прибавление первой единицы даст	0001
При прибавлении следующей единицы необходимо сделать перенос, что даст	0010
Прибавление следующей единицы даст	0011
При прибавлении следующей единицы снова необходимо сделать перенос во второй разряд (столбец) и еще один перенос в следующий разряд, что даст	0100

и т. д. Двоичные числа от 0 до 15 вместе с соответствующими десятичными числами представлены ниже

Десятичное число	Двоичное число	Десятичное число	Двоичное число
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

Заметим, что каждая двоичная цифра (двоичный разряд, или бит — bit) представляет собой целую степень двойки. Самая правая цифра (самый младший двоичный разряд) представляет собой $2^0=1$, следующая цифра $2^1=2$, далее: $2^2=4$, $2^3=8$, $2^4=16$, $2^5=32$, $2^6=64$, $2^7=128$ и т. д.

Так, число 1101 в двоичной системе счисления соответствует десятичному числу 13. Покажем это:

веса разрядов: 8421
двоичное число: 1101
Таким образом, имеем:

одну цифру 8:	$1\cdot8=8$
одну цифру 4:	$1\cdot4=4$
ни одной цифры 2:	$0\cdot2=0$
одну цифру 1:	$1\cdot1=1$

что и составляет число 13 в десятичной системе счисления.

Аналогично для числа 01100010:

веса разрядов	128	64	32	16	8	4	2	1
двоичное число	0	1	1	0	0	0	1	0

Следовательно, имеем:

ни одной цифры 128:	$0 \cdot 128 = 0$
одну цифру 64:	$1 \cdot 64 = 64$
одну цифру 32:	$1 \cdot 32 = 32$
ни одной цифры 16:	$0 \cdot 16 = 0$
ни одной цифры 8:	$0 \cdot 8 = 0$
ни одной цифры 4:	$0 \cdot 4 = 0$
одну цифру 2:	$1 \cdot 2 = 2$
ни одной цифры 1:	$0 \cdot 1 = 0$

что составляет число 98 в десятичной системе счисления.

Аппаратура проектируется обычно так, что положительное напряжение (U_+) воспринимается как 1, а нулевое — как 0 (положительная логика). Как было показано ранее, данные в аппаратуре представлены порциями, длина которых кратна восьми двоичным разрядам. Информация в такой форме подготовлена для одновременной передачи по шинам связи: 8 разрядов в шину данных и 16 — в адресную шину. Информация, кратная восьми двоичным разрядам, составляет слово. Восьмиразрядное слово называется байтом (byte), а 2^{10} байт (1024) составляют один килобайт (1 К).

Некоторые ранние модели микропроцессоров, такие как 4004 и 4040, имели всего 4-разрядные шины данных. Четырехразрядное слово называется полусловом, полубайтом или ниблом (nibble).

Рассмотрим, почему микропроцессор с 16-разрядной адресной шиной оказывается способным произвести адресацию информации до 64 К памяти. Шестнадцать адресных линий позволяют сформировать любой адрес от 0000 0000 0000 0000 до 1111 1111 1111 1111. В десятичной системе счисления это соответствует числам от 0 до 65 535. Другими словами, при наличии 16-разрядной шины существует (включая нулевой адрес) 65 536 различных адресов или возможных мест в памяти. Это соответствует объему памяти 64×1024 или 64 К. Заметим также, что 65 536 и есть 2^{16}. В общем случае n адресных линий (передачи адреса) позволяют обеспечить адресацию 2^n мест размещения информации (ячеек) в памяти.

Естественно, не все микро-ЭВМ обладают достаточной памятью для использования всех 65 536 возможных адресов.

Почему программируют в двоичном коде?

Аппаратура микропроцессора может воспринимать только два уровня напряжения: U_+ и 0, поэтому необходимо, чтобы все команды и данные были представлены в соответствующей форме, т. е. в двоичной (машинный язык).

Поскольку писать в двоичных кодах не совсем удобно, предпочитают писать программы в различных других кодах и пользуются специальными программами из ПЗУ для преобразования этих кодов в двоичные.

Зачем пользуются шестнадцатеричным кодом?

В некоторых случаях, когда действительно необходимо разобраться, что же происходит внутри микро-ЭВМ, лучше всего пользоваться шестнадцатеричным кодом, поскольку он позволит проследить за всеми единицами и нулями, избавив при этом от необходимости все время их писать.

В шестнадцатеричной системе счисления в качестве основания используется число 16 и, кроме десяти цифр, буквы латинского алфавита A, B, C, D, E, F для представления количественных эквивалентов 10, 11, 12, 13, 14, 15 соответственно.

Сравним различные представления чисел от 0 до 15:

Десятичное число	Двоичное число	Шестнадцатеричное число	Десятичное число	Двоичное число	Шестнадцатеричное число
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

Так, например, десятичное число 11 в двоичном коде записывается как 1011, а в шестнадцатеричном как B.

Чем же может помочь шестнадцатеричное представление чисел?
Оно же кажется еще более сложным?

Если число 1101001010111110 (попробуйте написать его на листе бумаги) потребовалось бы использовать в качестве элемента программы, то было бы гораздо целе-

сообразнее разбить его на 4-разрядные полуслова (ниблы):

1101 0010 1011 1110

а затем каждый фрагмент записать в шестнадцатеричном коде:

1101	0010	1011	1110
D	2	B	E

что дало бы D2BE.

Такой фрагмент программы гораздо легче записать безошибочно и при использовании кодов, представляющих собой операции, такую запись в шестнадцатеричном представлении проще запомнить, чем длинные последовательности 0 и 1. При желании всю программу можно сравнительно легко записать в шестнадцатеричном коде.

Как написать программу в шестнадцатеричном коде?

Сначала следует взять перечень того, что может делать микропроцессор, с указанием кодов операций, при помощи которых эти действия осуществляются. Такой перечень называется набором команд микропроцессора, он всегда прилагается при покупке микропроцессора.

Вот примеры некоторых команд для наиболее распространенной модели микропроцессора, приведенные вместе с шестнадцатеричными кодами операций, представляющими двоичные коды, заставляющие микропроцессор выполнять эти команды, т. е. организовывать правильные последовательности «подключений» к каналам передачи информации, к правильным местам ее хранения и обработки.

Назначение команд	Шестнадцатеричные коды
LOAD (ЗАГРУЗИТЬ) содержимое памяти по адресу, который будет указан сразу после этого кода операции, в определенное место (регистр) центрального процессора, называемое аккумулятором	AD
ADD (with carry) (СЛОЖИТЬ) (с учетом переноса) содержимое памяти по адресу, который будет указан сразу после кода этой операции, с содержимым аккумулятора и поместить затем ответ в аккумулятор вместо находящегося там числа	6D
STORE (ЗАПОМНИТЬ) содержимое аккумулятора в памяти по адресу, который будет указан после этого кода операции	8D

Продолжение

Назначение команд	Шестнадцатеричные коды
STOP (СТОП)	00
CLEAR (ОЧИСТИТЬ) те места в памяти центрального процессора, в которых хранятся переносы, оставшиеся от предыдущих вычислений (флаги переносов)	18

Попробуем написать реальную программу сложения двух чисел. Прежде чем начать писать ее в шестнадцатеричном коде, представим себе мысленно, что же должно произойти внутри микро-ЭВМ. Напомним, что микро-ЭВМ состоит из следующих четырех основных узлов: устройства ввода, устройства вывода, центрального микропроцессорного элемента и внешней памяти.

Будем считать, что центральный микропроцессорный элемент состоит из следующих трех основных блоков:

1) блока временного хранения информации, называемого регистрами;

2) арифметическо-логического устройства, или АЛУ (ALU — arithmetic and logic unit), выполняющего арифметические и логические операции подобно обычному калькулятору;

3) устройства управления, или УУ.

Представим себе, что внешняя помять — это некий стеллаж или книжный шкаф, состоящий из нумерованных отделений. Присвоенные отделениям номера являются адресами участков памяти, или адресами ячеек. Представленная такой моделью память и будет использоваться для хранения программ, данных и команд.

Управляющее устройство, как подсказывает его название, управляет всеми другими блоками и обеспечивает поочередное выполнение команд программы.

Что же должна выполнять программа сложения? Заметим, что числа, предназначенные для сложения, уже помещены в память, а именно в ячейки с номерами *102* и *203* (рис. 3.1, *а*).

Теперь необходимо исключить все переносы, оставшиеся от предыдущих вычислений. Затем нужно перенести копию данных, содержащихся в ячейке *102*, в аккумулятор, находящийся в АЛУ. Далее необходимо скопировать данные из ячейки *203* и перенести копию в АЛУ, для сложения с уже имеющимися там данными. Наконец, следует переписать копию результата в память, т. е. перенести ее в хранилище, например, в ячейку *101*.

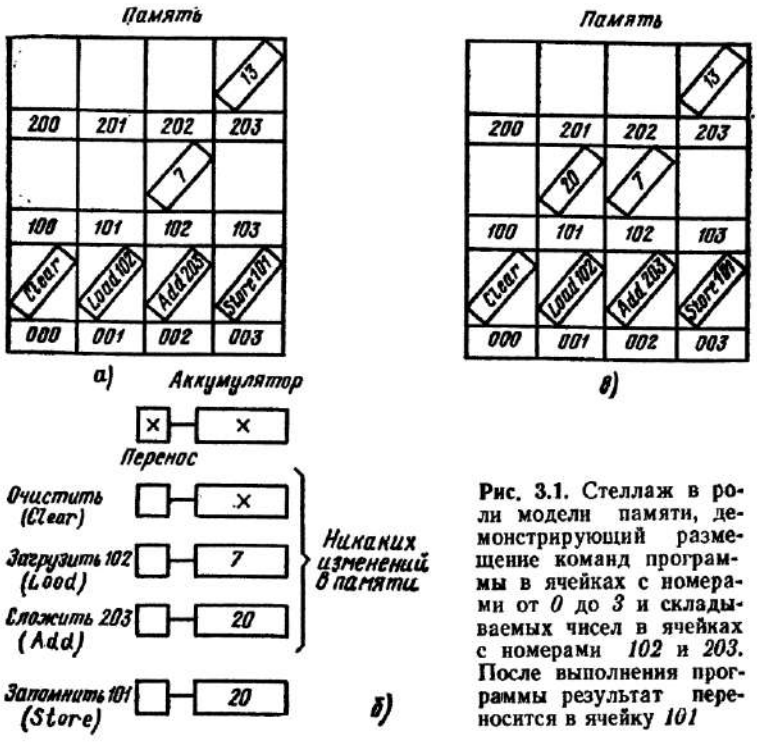


Рис. 3.1. Стеллаж в роли модели памяти, демонстрирующий размещение команд программы в ячейках с номерами от *0* до *3* и складываемых чисел в ячейках с номерами *102* и *203*. После выполнения программы результат переносится в ячейку *101*

Данные в памяти не разрушаются при считывании. Они просто копируются или переписываются. Данные остаются в памяти до тех пор, пока другое число не будет помещено на то же место или пока не будет выключено питающее напряжение.

Итак, командами в данном случае являются (рис. 3.1, *б*):

CLEAR (ОЧИСТИТЬ) флажок переноса;
LOAD (ЗАГРУЗИТЬ) (данные в ячейке) 102;
ADD (СЛОЖИТЬ) (данные в ячейке) 203;
STORE (ЗАПОМНИТЬ) (ответ в ячейке) 101.

Представленные в шестнадцатеричном коде для ввода в типичную микро-ЭВМ с шестнадцатеричной клавиатурой, эти команды выстроятся следующим образом.

Освободившись от переносов, оставшихся от предыдущих вычислений, получим:

18 в шестнадцатеричном коде.

Затем следует переписать (или произвести выборку из памяти) первое число и поместить его во временное запоминающее устройство (аккумулятор), находящееся внутри центрального микропроцессорного устройства. Таким образом, следующая команда в шестнадцатеричном коде — это AD, поэтому имеем:

```
18
AD
```

Однако эта команда требует указания адреса ячейки памяти, в которой хранятся необходимые данные. Пусть этот адрес будет 0066 в шестнадцатеричном коде (102 в десятичном). Этот адрес должен быть записан после кода операции СЛОЖИТЬ младшими значащими цифрами вперед, т. е. как 6600. (Пусть Вас не смущает, что шестнадцатеричное число записано наоборот, т. е. младший байт записан слева. Если младшие байты размещены первыми слева, команды, не содержащие более старших байтов, выполняются быстрее.) Итак, имеем:

```
18
AD66 00
```

Далее к этому числу следует прибавить число, содержащееся в другой ячейке памяти (например, в ячейке с адресом OOCB в шестнадцатеричном коде или 203 в десятичном). Итак, очередная команда есть 6D, за которой следует номер ячейки памяти с самым младшим байтом слева (CB 00)[1]:

```
18
AD66 00
6DCB 00
```

На следующем шаге необходимо переслать ответ, находящийся в аккумуляторе, обратно в память на вполне определенное место (например, в ячейку с адресом 0065 в шестнадцатеричном коде или 101 в десятичном). Поэтому следующей будет команда 8D с адресом 65 00, что даст:

```
18
AD66 00
6DCB 00
8D65 00
```

---

[1] Байту соответствуют два шестнадцатеричных разряда. Номер ячейки после кода команды записывается наоборот побайтно. Разряды внутри байта при этом не меняются местами. *(Прим. пер.)*

Таким образом, теперь имеется записанная программа, и ее следует ввести в микро-ЭВМ. Это производится путем записи команд в последовательные адреса памяти, после чего, когда микропроцессор получит управляющий сигнал начала работы, он сможет последовательно, шаг за шагом, считать всю программу.

Вернемся к рассматриваемой модели. Поместим команду CLEAR (ОЧИСТИТЬ) в ячейку с номером 000, команду LOAD 102 (ЗАГРУЗИТЬ 102) — в ячейку с адресом 001, команду ADD 203 (СЛОЖИТЬ 203)—в ячейку 002 и, наконец, команду STORE 101 (ЗАПОМНИТЬ 101) — в ячейку 003 (рис. 3.1, *в*).

Естественно, нужно, кроме того, ввести данные (складываемые числа) в те места памяти, номера ячеек которых указаны в командах программы, т. е. в данном случае в ячейки 102 и 203. Это также легко сделать в шестнадцатеричном коде. Как показано выше, программа имеет следующий вид:

```
18
AD66 00
6DCB 00
8D65 00
```

В дальшейшем увидим, что каждая ячейка памяти в микро-ЭВМ может хранить только восемь двоичных разрядов. Но, как было сказано ранее, каждая пара шестнадцатеричных цифр как раз и представляет собой восемь двоичных разрядов. Например, число 18 в шестнадцатеричном коде представляет собой следующие двоичные числа:

```
          1               8
0 0 0 1       1 0 0 0
```

или 8-разрядное двоичное слово 00011000. Поэтому первую команду можно поместить в одну ячейку памяти, например в ячейку с номером 000 в десятичном коде, что соответствует значению 0000 в шестнадцатеричном коде. Однако следующая команда в шестнадцатеричном коде, а именно код операции AD и шестнадцатеричный адрес 6600, потребует для своего размещения трех ячеек памяти: 0001, 0002 и 0003 (рис. 3.2). Очередная команда 6D CB 00 потребует для размещения следующих трех ячеек памяти: 0004, 0005 и 0006, и, наконец, последняя команда 8D 65 00 потребует следующих трех ячеек: 0007, 0008 и 0009 (все номера в шестнадцатеричном коде).

Затем необходимо добавить команду STOP (СТОП), которая, как видно из набора команд, приведенного выше, имеет шестнадцатеричный код 00. Ее следует поместить в следующую ячейку памяти с шестнадцатеричным номером 000А.

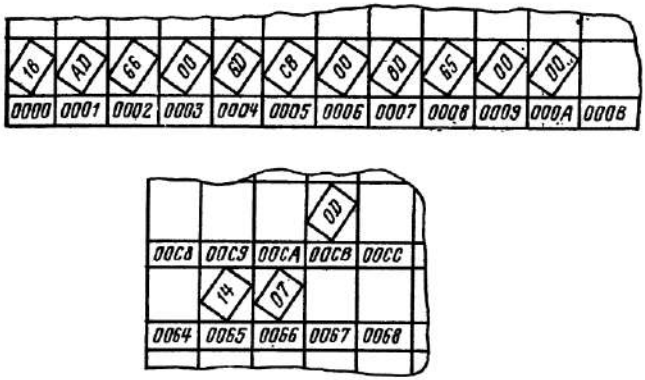


Рис. 3.2. Модель памяти с шестнадцатеричными кодами в виде стеллажа. Два числа, которые нужно сложить, 07 и 0D (в шестнадцатеричной системе счисления) следует поместить в ячейки 0066 и 00CB. Готовый ответ — в ячейке 0065 (заметим, что шестнадцатеричное число 14 соответствует десятичному числу 20)

Два десятичных числа, которые нужно сложить, например 7 (шестнадцатеричное 07) и 13 (шестнадцатеричное 0D), следует записать в ячейки памяти, указанные в программе, т. е. в ячейки с шестнадцатеричными номерами 0066 и 00CB. Затем можно передать управление монитору микро-ЭВМ нажатием ключа RUN или GO (ПУСК) (чаще всего после выполнения других шагов или этапов управления, характерных для каждой отдельной микро-ЭВМ, например выдачи указания, где можно найти начало программы). Чтобы проследить дальнейший ход событий, вернемся снова к рассматриваемой модели.

Схемы управления содержат программный счетчик, или счетчик команд, необходимый для того, чтобы управляющее устройство могло осуществить выбор первой команды из первой ячейки памяти, выполнить ее, затем осуществить выбор и выполнить вторую команду, затем третью и т. д., пока не будет достигнут конец програм-

мы, где ее работа прекратится. Если после этого прочитать содержимое ячейки 101, то там будет найден ответ.

Чтобы выполнить сложение двух других чисел, нужно изменить только данные в ячейках 102 и 203 и запустить программу снова.

Если записать адреса ячеек памяти, где хранятся команды и коды необходимых операций, а также адреса, на которые в них имеются ссылки, получится следующая картина:

Адреса ячеек памяти, в которых хранятся команды	Наименования операций	Адреса, на которые есть ссылка в командах
000	CLEAR	—
001	LOAD	ячейка 102
002	ADD	ячейка 203
003	STORE	ячейка 101
004	STOP	—

Для рассматриваемой программы в шестнадцатеричных кодах получим:

Адреса ячеек памяти, в которых хранятся коды операций	Коды операций	Адреса, на которые есть ссылка в командах (аргументы)	
0000	18	—	—
0001	AD	66	00
0004	6D	CB	00
0007	8D	65	00
000A	00	—	—

Далее объясним, каким образом программа, подобная рассмотренной, вызывается из памяти и используется для управления работой различных частей микро-ЭВМ. Тем, у кого микро-ЭВМ имеют только шестнадцатеричную клавиатуру ввода, программы приходится писать именно таким образом.

Зачем применяется программа ассемблера?

С увеличением длины программы все труднее становится запоминать коды различных операций и тем более списки адресов хранимых или вычисляемых данных. Если во время написания программы или потом, когда программа уже написана, обнаружится, что ее желательно изменить или отредактировать, необходимая при этом сортировка всех адресов заново может стать настоящим мучением.

Поэтому намного легче писать программу на символическом языке ассемблера. Когда она готова и введена

в машину, программа ассемблера преобразовывает все символические коды и адреса в программу на языке машины, размещает получившуюся программу на машинном языке в правильном порядке в памяти и берет на себя также заботу о правильном использовании всех адресов при прохождении программы.

Программа ассемблера может извлекаться из ПЗУ или вводиться с магнитной ленты путем считывания в ОЗУ микро-ЭВМ (в память с произвольным доступом) каждый раз, как только она понадобиться. Естественно, можно также написать свою собственную программу ассемблера.

Зачем пользуются языками высокого уровня?

С языками высокого уровня, такими как БЕЙСИК, еще меньше проблем, поскольку не нужно заботиться об истинных адресах ячеек памяти. Программа компилятор или интерпретатор возьмет всю заботу о них на себя, а также даст возможность использовать для написанных команд коды (выражения), имеющие вид английских слов. Более того, некоторые команды могут быть сгруппированы вместе как некое выражение (утверждение) форматом в одну строчку.

На языке БЕЙСИК, например, сложение двух чисел имеет вид простого выражения:

LET C = A + B

Искомый ответ можно получить на экране дисплея, не зная даже, где он находится в памяти, используя команду

PRINT C

Полная программа, включая ввод данных и печать результата, будет выглядеть следующим образом:

```
10 LET    A=7
20 LET    B=13
30 LET    C=A+B
40 PRINT  C
50 END
```

Примечание. Строчки программы принято нумеровать целыми десятками, чтобы легко было вернуться и вставить пропущенную строчку, если вы ошиблись.

Вся эта программа вводится с помощью клавиатуры и выполняется после набора команды RUN.

Почти сразу в следующей строчке на экране дисплея появляется значение C, равное 20.

Ну и что? Этот же результат можно получить еще быстрее, сложив 7 и 13 в уме!

Да, но простые программы, рассматривавшиеся до сих пор, имели целью лишь проиллюстрировать, как следует размещать данные в памяти и затем пересылать и обрабатывать их, как это требуется. Это может проделать даже простой калькулятор! Микро-ЭВМ как более мощное вычислительное средство отличает ее способность выполнять не только обычный набор арифметических операций, но и более сложные команды, такие как логические, условные, а также команды сравнения, что существенно расширяет возможности составления и реализации более сложных программ.

ГЛАВА ЧЕТВЕРТАЯ

Основные логические операции

Если микро-ЭВМ — всего лишь совокупность переключателей, то как она может выполнять арифметические и логические операции и операции сравнения?

Логические операции наиболее легко реализуются. Они основаны на булевой алгебре, предложеннной в 1854 г. Дж. Булем для исследования процессов человеческих рассуждений. Позднее аппарат булевой алгебры стал применяться для анализа электрических переключательных схем. Существует пять основных, наиболее часто используемых операций: НЕ (NOT), И (AND), ИЛИ (OR), И-НЕ (NAND), ИЛИ-НЕ (NOR). Каждая из них может реализоваться путем построения соответствующей электрической схемы, называемой логическим элементом. Различные логические элементы, по-разному соединенные, образуют схемы, реализующие довольно сложные функции, такие как сравнение, вычитание, сложение и т. п.

Рассмотрим эти пять основных логических операций с тем, чтобы проследить, как их можно реализовать с помощью простых переключателей. Не пытайтесь пока преобразовать их в транзисторные схемы. После освоения этих логических операций, пользуясь несколькими

существующими типовыми приемами, нетрудно будет получить транзисторную схему, реализующую любую операцию, с помощью всего одной базовой КМОП-структуры.

Что такое элемент НЕ?

Операция взятия дополнения до единицы, или операция инверсии, или операция НЕ (NOT), обозначается черточкой над инвертируемой величиной. Так, например, «НЕ A» записывается как $\bar{A}$.

Рассматриваемая аппаратура способна воспринимать лишь двухуровневые сигналы, соответствующие двум ло-

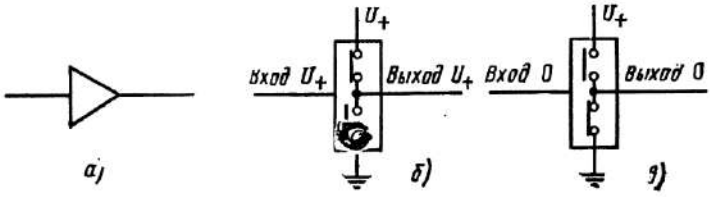


Рис. 4.1. Элемент НЕ:

а — схемное обозначение; *б* — инверторный переключатель (вход 0, выход 1); *в* — инверторный переключатель (вход 1, выход 0)

гическим состояниям: 1, представляемой положительным потенциалом U_+, и 0, представляемым нулевым потенциалом (в положительной логике). Следовательно, если A есть 0, НЕ A должно быть 1, и наоборот. Схемное обозначение элемента НЕ (инвертора) приведено на рис. 4.1, *а*.

Реализацию операции НЕ можно проиллюстрировать с помощью рассмотренной ранее ключевой модели, т. е. двух ключей в одном корпусе, подключающих выходную цепь либо к источнику положительного напряжения U_+, либо к земле. Если входной сигнал равен нулю, выход переключателя соединяется с полюсом U_+ (логическая 1; рис. 4.1, *б*). Если входной сигнал равен U_+, выходной сигнал будет нулевым (рис. 4.1, *в*).

Все схемы логических элементов, рассматриваемых в этой главе, обладают одной общей особенностью: значение сигнала на выходе каждой схемы зависит только от значений сигналов, поданных в настоящий момент времени на все ее входы, и меняется с изменением этих

входных сигналов. Схемы с такой особенностью относят к схемам комбинационной логики, или к комбинационным схемам.

Для комбинационных элементов можно составить таблицы соответствия выходов входам, называемые таблицами истинности логических элементов, чтобы более

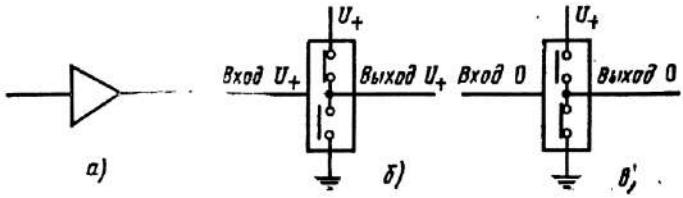


Рис. 4.2. Буфер:

а — схемное обозначение; *б* — инверторный переключатель (вход 1, выход 1); *в* — инверторный переключатель (вход 0, выход 0)

четко представить себе те логические операции, которые могут реализовываться с их помощью.

Таблица для операций НЕ будет иметь следующий вид:

Таблица соответствия выхода и входа

Вход	Выход
0	U_+
U_+	0

Таблица истинности

A	$\overline{A}$
0	1
1	0

Таблица соответствия входных и выходных сигналов получается простым перечислением (перебором) всех возможных комбинаций значений входных сигналов и последующей записью рядом с каждой из комбинаций соответствующего значения выходного сигнала.

Элемент, который встречается довольно часто, реализует самую простую операцию. Этим элементом является буфер. Схемное обозначение буфера приведено на рис. 4.2, *а*. Он реализует операцию, противоположную операции НЕ[1], и, кроме того, обеспечивает электрическое согласование входа и выхода.

[1] Функцию повторения (т. е. функцию, повторяющую значения входных сигналов). *(Прим. ред.)*

Работу буфера можно также проиллюстрировать с помощью ключевой модели, которая рассматривалась при описании инвертора. Однако в данном случае, когда входной сигнал равен U_+, на выходе также будет U_+. Если входной сигнал равен 0, на выходе также будет 0. Таблица истинности для буфера выглядит следующим образом:

Вход	Выход
0	0
1	1

Что такое элемент И ?

Операция И (AND) обозначается точкой (как обычное умножение), которая при чтении воспринимается как союз «и», т. е. запись $A \cdot B$ читается как A и B[1].

В схеме И сигнал на выходе элемента, соответствующий логической 1, появляется только в том случае, когда аналогичные сигналы присутствуют одновременно на входах A и B. Схемное обозначение элемента И приведено на рис. 4.3, *а*.

Ключевую модель элемента, реализующего операцию И, можно рассматривать как модель, состоящую из совмещенных в одном корпусе буферов A и B, включенных таким образом, что одни их коммутирующие элементы A и B включены последовательно, один за другим (в цепи между положительным полюсом U_+ и выходом), а другие — параллельно (в цепи между землей и выходом) (рис. 4.3). Заметим, что, поскольку каждый инвертирующий ключ выполняет здесь роль буфера, при нулевом сигнале на любом из входов замыкается соответствующий контакт в цепи между землей и выходом и размыкается цепь между выходом и полюсом U_+, обеспечивая тем самым нулевой сигнал на выходе (рис. 4.3, *б*).

Если на вход A поступил сигнал U_+, а на вход B сигнал, равный 0, то сигнал на выходе остается нулевым, хотя верхний контакт замкнется, а один из нижних — разомкнется (рис. 4.3, *в*). Аналогичное происходит, если

[1] Иногда используется специальный знак & или реже ∧. (*Прим. пер.*)

на вход B поступит сигнал U_+, а на входе A он нулевой (рис. 4.3, *г*). Однако если сигнал U_+ поступит на входы A **И** B одновременно, то на выходе схемы будет сформирован сигнал U_+ (рис. 4.3, *д*). Отсюда и название «опе-

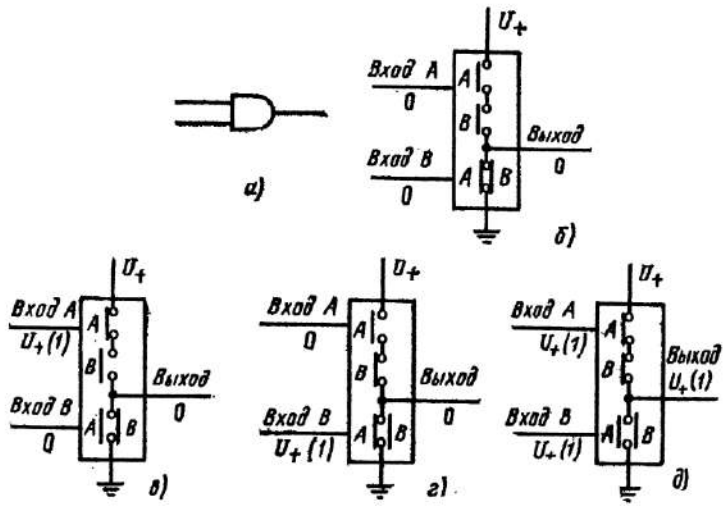


Рис. 4.3. Элемент И:

а — схемное обозначение; *б—д* — инверторный переключатель для четырех возможных комбинаций входных сигналов

рации **И**». Таблица истинности для операции И имеет следующий вид:

A	B	A И B
0	0	0
0	1	0
1	0	0
1	1	1

Что такое элемент ИЛИ?

Операция ИЛИ (OR) обозначается знаком +, который[1] при чтении соответствующих формальных выраже-

[1] Чаще для обозначения этой операции используется специальный знак $\vee$. *(Прим. пер.)*

ний воспринимается как союз «или». В булевой алгебре $A+B$ означает A ИЛИ B.

В схеме ИЛИ сигнал на выходе элемента, соответствующий логической 1, появляется в случае присутствия

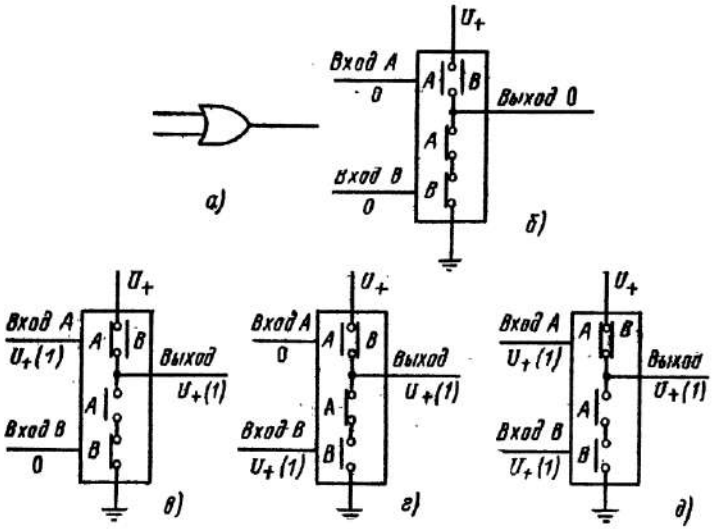


Рис. 4.4. Элемент ИЛИ:

а — схемное обозначение; *б—д* — инверторный переключатель для четырех возможных комбинаций входных сигналов

единичного сигнала хотя бы на одном из входов: A или B (либо на том и на другом одновременно). Схемное обозначение элемента ИЛИ приведено на рис. 4.4, *а*.

Ключевую модель элемента, реализующего операцию ИЛИ, также можно рассматривать как состоящую из совмещенных в одном корпусе буферов A и B. Однако теперь их коммутирующие элементы A и B будут включены параллельно в цепи между полюсом U_+ и выходом элемента и последовательно один за другим в цепи между землей и выходом (рис. 4.4).

Если на вход A **ИЛИ** B поступит сигнал U_+, выход через соответствующий параллельный контакт соединится с полюсом U_+. Заметим, что сигнал на выходе сохранит значение U_+ и в том случае, если сигналы U_+ поступят одновременно на входы A и B.

Отсюда следует и название «операция **ИЛИ**», поскольку выход получает значение U_+, если на входах *A* **ИЛИ** *B* поочередно или одновременно появляется сигнал U_+. Таблица истинности для операции ИЛИ имеет следующий вид:

A	*B*	*A* ИЛИ *B*
0	0	0
0	1	1
1	0	1
1	1	1

Что такое элемент И-НЕ?

Операция И-НЕ (NAND, соответствует NOT AND) обозначается следующим образом: $\overline{A \cdot B}$. На выходе схемы И-НЕ сигнал U_+ отсутствует только в том случае, когда сигналы U_+ имеются одновременно на входах *A* и *B*. Схемное обозначение элемента И-НЕ приведено на рис. 4.5, *а*.

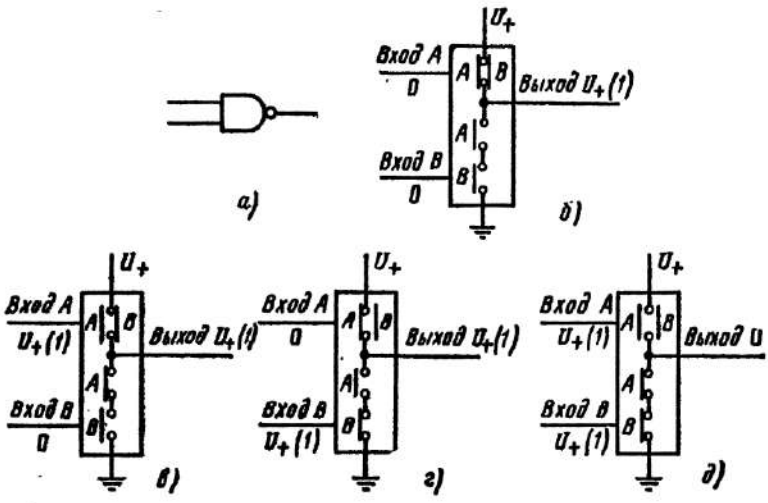


Рис. 4.5. Элемент И-НЕ:

а — схемное обозначение; *б—д* — инверторный переключатель для четырех возможных комбинаций входных сигналов

Ключевую модель элемента, реализующего операцию И-НЕ, можно рассматривать как состоящую из совмещенных в одном корпусе инверторов A и B, включенных таким образом, что их коммутирующие элементы A и B включены параллельно в цепи между полюсом U_+ и выходом элемента и последовательно, один за другим, в цепи между землей и выходом (рис. 4.5). Заметим, что поскольку в данном случае в качестве ключей используются инверторы, нулевой входной сигнал каждого инвертора приведет к замыканию цепи между полюсом U_+ и выходом элемента. В соответствующей модели элемента ИЛИ использовалась точно такая же схема соединения ключей, однако в качестве ключей использовались буфера.

Итак, если сигнал U_+ поступит только на один из входов элемента A или B, на выходе элемента И-НЕ сохранится сигнал U_+ (за счет воздействия нулевого сигнала на другом входе). Если сигналы U_+ поданы одновременно на входы A и B, выход элемента И-НЕ согласно схеме на рис. 4.5, *д* соединится с землей. Отсюда и следует название «операция НЕ (A И B)» или «операция И-НЕ»[1]. Таблица истинности для операции И-НЕ выглядит следующим образом (сравните с соответствующей таблицей для операции И):

A	B	A И-НЕ B
0	0	1
0	1	1
1	0	1
1	1	0

Что такое элемент ИЛИ-НЕ?

Операция ИЛИ-НЕ (NOR, соответствует NOT OR) записывается так $\overline{A+B}$ и означает НЕ (A ИЛИ B). Другими словами, на выходе схемы ИЛИ-НЕ отсутствует сиг-

[1] Некоторая нелогичность такого объяснения вытекает из общепринятого в отечественной литературе и, в частности, в ГОСТ наименования для этой операции «И-НЕ» вместо «НЕ-И», употребляемого в англоязычной литературе. С точки зрения записи реализуемой функции, эту операцию правильнее именовать «НЕ-И», а с точки зрения схемного обозначения соответствующего логического элемента, «И-НЕ» (см. рис. 4.5, *а*). *(Прим. пер.)*

нал U_+ во всех случаях, когда такой сигнал имеется хотя бы на одном входе. Схемное обозначение элемента ИЛИ-НЕ приведено на рис. 4.6, *а*.

Ключевую модель элемента, реализующую операцию ИЛИ-НЕ, можно также рассматривать как состоящую из совмещенных в одном корпусе инверторов *A* и *B*, однако в этом случае их коммутирующие элементы *A* и *B* будут включены последовательно один за другим в цепи меж-

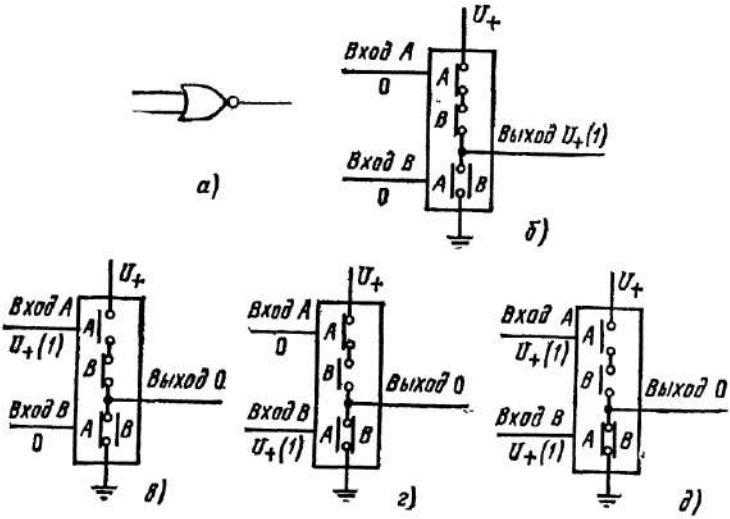


Рис. 4.6. Элемент ИЛИ-НЕ:

а — схемное обозначение; *б—д* — инверторный переключатель для четырех возможных комбинаций входных сигналов

ду полюсом U_+ и выходом элемента и параллельно в цепи между землей и выходом (рис. 4.6). Отметим, что, как и в предыдущем случае, переключатели являются инверторами, поэтому при поступлении нулевого сигнала на любой вход соответствующий контакт замыкается в цепи между полюсом U_+ и выходом и размыкается в цепи между землей и выходом. Именно это обстоятельство отличает работу данной модели от работы модели элемента И, в которой используется та же схема соединения ключей, однако в качестве последних используются не инверторы, а буфера.

Если сигнал U_+ поступит на входы A и B (на один из них или на оба одновременно), то выход элемента получит потенциал земли. Отсюда название «операция НЕ (A ИЛИ B)», или «операция ИЛИ-НЕ»[1]. Таблица истинности для операции ИЛИ-НЕ имеет следующий вид (сравните с таблицей для элемента ИЛИ):

A	B	A ИЛИ-НЕ B
1	0	1
0	1	0
0	0	0
1	1	0

Операций такое множество, что в них легко запутаться. Как бы все это упростить?

Это можно сделать при условии, что вы хорошо разобрались в их логических возможностях. На самом деле нет особой необходимости отводить отдельный элемент или отдельную электронную схему для выполнения каждой отдельной логической операции.

Высокая степень интеграции элементов стала возможной благодаря производству большого числа однотипных базовых компонентов, повторяющихся в нескольких конфигурациях много раз подряд на одном кристалле. Один из путей упрощения разработки электронных логических схем состоит в использовании инвертора в качестве основного «строительного» блока. Как было показано ранее, с помощью двух инверторов[2] могут строиться как схемы И-НЕ, так и схемы ИЛИ-НЕ. Путем соединения элементов И-НЕ, ИЛИ-НЕ и инверторов в раз-

[1] Случай, аналогичный объяснению названия операции И-НЕ (см. предыдущую сноску). *(Прим. пер.)*

[2] Здесь имеется в виду не собственно инвертор как элемент НЕ, а его ключевая модель или рассмотренный ранее так называемый инверторный переключатель, который действительно может служить основой для построения элементов И-НЕ, ИЛИ-НЕ. Однако отсюда не следует, что с помощью только операции НЕ можно выразить операции И-НЕ, ИЛИ-НЕ. В данном случае при построении ключевых моделей И-НЕ, ИЛИ-НЕ использовались последовательные и параллельные варианты включения контактов переключателей; при этом последовательное соединение контактов A и B соответствует выполнению на них операции И, а параллельное — операции ИЛИ. Поэтому при построении ключевых моделей элементов И-НЕ, ИЛИ-НЕ, строго говоря, использовались три операции: НЕ, И, ИЛИ. *(Прим. ред.)*

личных сочетаниях можно строить схемы, реализующие любые другие логические операции[1].

Итак, фактически любые логические операции могут реализоваться с помощью микросхемы, построенной с применением правильного набора «базовых инверторов».

На практике все логические операции и реализуются путем использования любых подходящих для этого операций логических элементов. При этом операция, выбранная в качестве базовой, должна обеспечивать возможность построения наиболее простым и дешевым способом соответствующего логического элемента и схем с его применением. На выбор базовой операции существенное влияние оказывает технология производства и тип логических структур (ДТЛ, ТТЛ, КМОП и т. п.). Например, в КМОП-структурах основным «строительным» блоком является базовый инвертор. Два инвертора, соединенные последовательно, взаимно нейтрализуют инвертирование сигнала и образуют буфер.

Если выход элемента И-НЕ, построенного в свою очередь из инверторов, инвертирован, то в результате происходит выполнение операции И.

Если выход элемента ИЛИ-НЕ, построенного также из инверторов, инвертирован, то такая схема реализует операцию ИЛИ.

Во времена, предшествовавшие появлению высокой степени интеграции элементов и разработке микропроцессоров на одном кристалле, приходилось строить сложные логические схемы из набора небольших микросхем. Они все еще применяются иногда для комплектации цепей и схем поддержки для микропроцессоров. Каждая такая микросхема обычно содержит от четырех до шести однотипных схем. Например, микросхема, состоящая из четырех схем И-НЕ, или микросхема, состоящая из четырех схем ИЛИ, или из шести буферов и т. д. К числу таких наборов микросхем принадлежат микросхемы серии 7400 (т. е. 7400, 7401, 7402 и т. д.), выполненные по ТТЛ-технологии, а также серий 4000 и 74С, выполненные по КМОП-технологии.

---

[1] Для построения схемы, реализующей любую логическую операцию, вполне можно обойтись всего одним типом элементов: либо И-НЕ, либо ИЛИ-НЕ; правда, в этом случае схемы, возможно, окажутся сложнее (потребуется большее число элементов), чем при использовании двух типов элементов в сочетании с инверторами *(Прим. ред.)*

Самые дешевые в изготовлении и, следовательно, наиболее доступные по цене при покупке элементы — это элементы И-НЕ и инверторы. Поэтому очень часто при построении сложных схем на базе микросхем используются микросхемы с элементами И-НЕ, с помощью которых и реализуются любые требуемые функции. Способ построения схем из одних и тех же элементов удобен еще и тем, что в ряде случаев при необходимости расширения функциональных возможностей схемы удается использовать не задействованные сразу (запасные) элементы некоторых микросхем.

Элемент И-НЕ с обычными входами можно использовать в качестве инвертора. Элемент И можно получить инвертированием выхода элемента И-НЕ. Элемент ИЛИ получается инвертированием входов элемента И-НЕ, а элемент ИЛИ-НЕ — инвертированием выхода полученного элемента ИЛИ.

Чтобы убедиться в том, что Вы действительно разобрались в рассматриваемых логических операциях, было бы неплохо проверить все эти и предыдущие логические соотношения путем построения соответствующих ключевых моделей КМОП-структур и последующего анализа их работоспособности.

Как можно проверить правильность логических операций?

Это можно сделать путем построения таблицы истинности, соответствующей работе исследуемой схемы. Проанализируем возможность выполнения операции ИЛИ схемой, составленной из одного элемента И-НЕ и двух инверторов, подключенных к его входам (рис. 4.7).

Когда на оба внешних входа схемы поступят нулевые сигналы, на ее выходе будет также нулевой сигнал, поскольку инверторы изменят внешние нулевые сигналы на U_+, являющиеся входными сигналами элемента И-НЕ (рис. 4.7, *в*).

Если сигнал U_+ поступит на вход A или B (рис. 4.7, *г*–*е*) (на один из них или на оба одновременно), на соответствующих входах элемента И-НЕ будут нулевые сигналы (по крайней мере, на одном из входов в любом из рассматриваемых здесь случаев), что вызовет появление сигнала U_+ на выходе схемы. Тогда таблица истин-

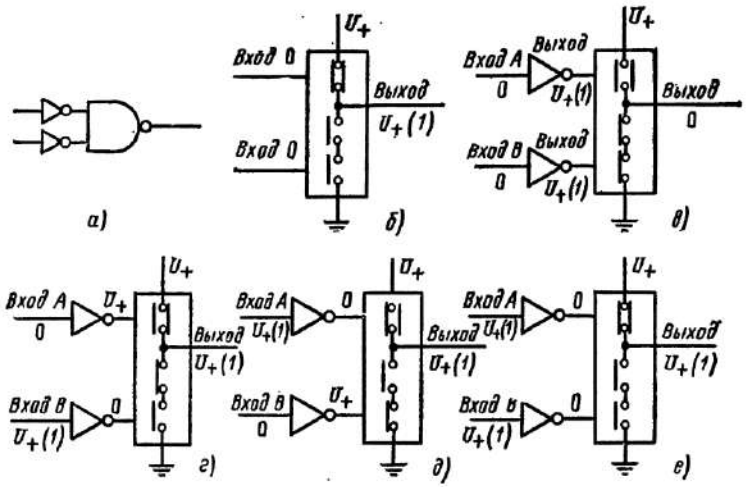


Рис. 4.7. Реализация операции ИЛИ схемой из двух элементов НЕ и одного элемента И-НЕ:

а — схемное обозначение; *б* — переключатель, имитирующий работу элемента И-НЕ; *в—е* — инверторный переключатель с элементами НЕ на входах для четырех возможных комбинаций входных сигналов

ности для схемы на рис. 4.7, *а* будет иметь следующий вид:

A	*B*	Выход
0	0	0
0	1	1
1	0	1
1	1	1

Нетрудно заметить, что эта таблица совпадает с таблицей истинности для элемента ИЛИ.

Попытайтесь теперь сами проверить правильность работы других схем.

Функции переключателей в микропроцессорах выполняют транзисторы. Как же в действительности построены логические схемы?

Да, действительно, эти функции выполняют транзисторы. В микросхемах, построенных с применением МОП-технологии, это либо *p*-МОП-транзисторы, либо *n*-МОП-транзисторы, или и те и другие вместе: КМОП-структуры. Простые ключевые модели логических операций не-

трудно превратить в реальные КМОП-транзисторные элементы схем (напомним, что КМОП означает совместное использование в паре *p*-канального и *n*-канального МОП-транзисторов).

На рис. 4.8 приведены реальные схемы КМОП-структур трех элементов: НЕ, И-НЕ, ИЛИ-НЕ, ключевые модели которых были рассмотрены ранее.

Необходимо научиться различать особенности переключения двух транзисторов в таких схемах. Напомним, что их входы соединены между собой таким образом, что,

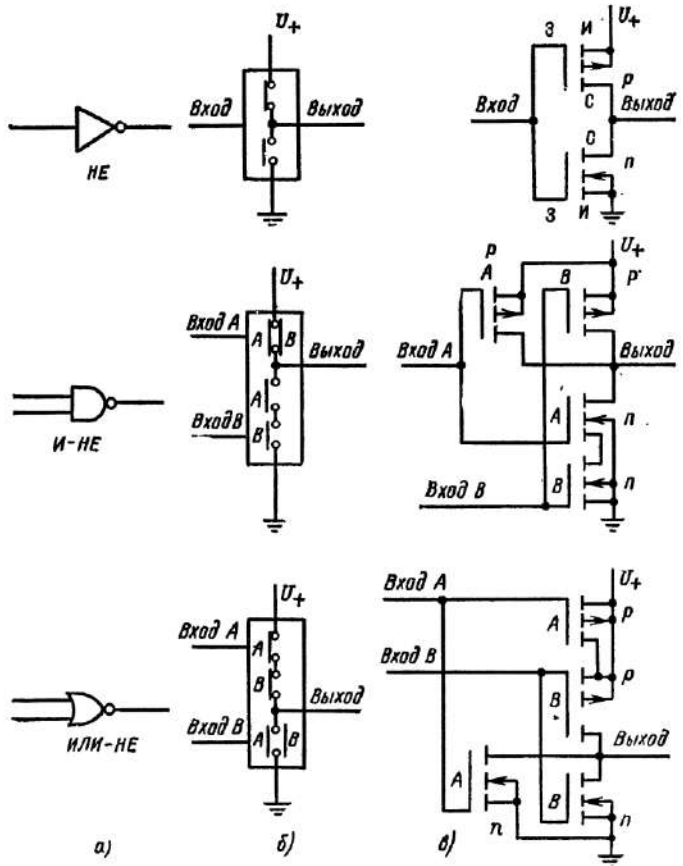


Рис. 4.8. Реальные схемы элементов:

а — схемное обозначение; *б* — ключевая модель; *в* — реальная схема КМОП-структуры

когда один транзистор включен, другой должен быть выключен. Например, если в схеме НЕ входы транзисторов соединены с положительным полюсом источника питания (на входы поступает сигнал U_+), то n-канальный МОП-транзистор будет открыт, а p-канальный МОП-транзистор заперт, в результате чего выход схемы будет соединен с землей (получит нулевой потенциал). Если входы транзисторов связаны с землей (на входах нулевой потенциал), то n-канальный МОП-транзистор будет заперт, а p-канальный — открыт, в результате чего выход схемы получит положительный потенциал U_+.

Эта схема — основной фрагмент КМОП-структур, и подобные дополняющие пары полевых транзисторов будут в дальнейшем неоднократно встречаться в более сложных схемах, построенных с применением КМОП-технологии.

Обратим внимание на элемент И-НЕ (рис. 4.8). Здесь тоже важно уметь различать особенности переключения транзисторов. В данной схеме четыре транзистора: по одному p-канальному и n-канальному для каждого входа. Возьмите лист бумаги и отметьте, какие транзисторы открыты, а какие закрыты для всех возможных значений входных сигналов. Определите значения соответствующих выходных сигналов и сравните результаты со значениями таблицы истинности для операции И-НЕ. Они должны совпасть. Проделайте то же самое для схемы ИЛИ-НЕ.

Не забывайте, что схема ИЛИ может быть получена путем размещения инвертора (НЕ) после элемента ИЛИ-НЕ (на пути распространения сигнала), что равносильно добавлению в схему ИЛИ-НЕ двух вспомогательных транзисторов. Точно так же можно получить схему И, разместив инвертор (НЕ) после элемента И-НЕ (или два вспомогательных транзистора). Буфер, как говорилось ранее, —это два последовательно включенных инвертора (НЕ).

Если Вы знакомились с литературой по КМОП-схемам, то, наверное, обратили внимание на то, что иногда в схемы включаются диоды. Они принципиально не влияют на работу схемы и служат только для ее защиты. Слой окисла между затвором и каналом в полевом транзисторе очень тонкий и может легко разрушиться под воздействием приложенного (свыше 60 В) напряжения. Добавленные в схему диоды образуют дополнитель-

ный запасной путь для прохождения зарядов и тем самым предохраняют полевой транзистор от выхода из строя. Но даже и при этом следует быть очень аккуратным при обращении с микросхемами на базе МОП-технологии. Для них опасны заряды статического электричества. Накопившийся на Вас от шерстяной или синтетической одежды либо от ковра заряд, попадая через руки на контакты микросхемы, с которой Вы работаете, может вывести ее из строя. Желательно использовать свободные от зарядов статического электричества (заземленную) поверхность и инструмент при работе с приборами, изготовленными на базе МОП-технологии. Нельзя дотрагиваться до выводов микросхемы, предварительно не разрядив себя и используемый в работе инструмент.

Как же все-таки используются все эти логические элементы?

Логические элементы используются всюду в микроЭВМ при выполнении логических и арифметических операций, а также при управлении потоком данных.

Однако прежде чем переходить к изложению особенностей их использования, рассмотрим еще одну схему, после чего сразу станет ясно, как можно произвести сравнение двух чисел, чтобы выяснить, имеем ли мы дело с одной и той же величиной или нет.

Вводимый новый элемент носит название ИСКЛЮЧАЮЩЕЕ ИЛИ (EXCLUSIVE OR или XOR). Этот элемент аналогичен элементу ИЛИ, но в отличие от последнего не формирует на выходе сигнала логической 1, если на оба его входа одновременно поступает положительный потенциал U_+. Другими словами, элемент фиксирует наличие сигнала 1 ИСКЛЮЧИТЕЛЬНО на одном из входов, но не на обоих одновременно. Схемное обозначение элемента ИСКЛЮЧАЮЩЕЕ ИЛИ[1] приведено на рис. 4.9, *а*, а его ключевая модель — на рис. 4.9, *б*. Нетрудно заметить, что соответствующую КМОП-структуру можно построить на основе ключевой модели, состоящей из трех инверторных переключателей.

Если $A=0$ и $B=0$, то $X=\overline{A}=U_+$, $Y=X=U_+$, и поэтому на выходе сигнал равен 0.

[1] Другое распространенное название — «сложение по модулю 2». (*Прим. пер.*)

Если $A=U_+$ и $B=0$, то $X=\overline{A}=0$, $Y=X=0$, и поэтому на выходе сигнал равен U_+.

Если $A=0$ и $B=U_+$, то $X=U_+$, но $Y=A=0$, и поэтому на выходе сигнал равен U_+.

Если $A=U_+$ и $B=U_+$, то $X=0$, но $Y=A=U_+$, и поэтому на выходе сигнал равен 0.

Таблица истинности для этой операции примет следующий вид:

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Операция ИСКЛЮЧАЮЩЕЕ ИЛИ обозначается знаком $\oplus$. Так, запись $A \oplus B$ означает либо A, либо B, но не оба вместе.

Если выполнить операцию НЕ над операцией ИСКЛЮЧАЮЩЕЕ ИЛИ, то можно получить следующую таблицу истинности:

A	B	Выход
0	0	1
0	1	0
1	0	0
1	1	1

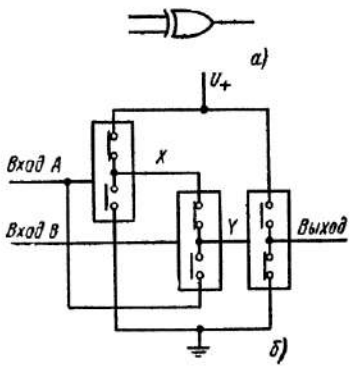


Рис. 4.9. Элемент ИСКЛЮЧАЮЩЕЕ ИЛИ:

а — схемное обозначение; *б* — ключевая модель

Заметим, что на выходе логического элемента, реализующего эту операцию, сигнал U_+ появляется только в том случае, если сигналы на двух его входах одинаковые. С помощью этой операции можно сравнивать два одноразрядных двоичных числа. Этот элемент называется компаратором (comparator), а операция — операцией СОВПАДЕНИЯ (COINCIDENCE)[1],

[1] Более распространенное название этой операции «ЛОГИЧЕСКАЯ РАВНОЗНАЧНОСТЬ», или «ЭКВИВАЛЕНТНОСТЬ», которое и используется далее. Название, которое дает автор, менее употребительно, его не нужно путать с понятием «схема совпадения», под которым подразумевается схема И. Операция ЭКВИВАЛЕНТНОСТЬ обозначается чаще знаком $\backsim$. *(Прим. ред.)*

обозначаемой символом $\odot$. Запись $A \odot B$ означает операцию «*A* ЭКВИВАЛЕНТНО *B*». Чтобы произвести сравнение одного восьмиразрядного двоичного числа с другим, потребуется восемь таких элементов, по одному на каждый разряд или линию данных

ГЛАВА ПЯТАЯ

Основные арифметические операции

Как микропроцессор выполняет арифметические операции?

Прежде чем рассматривать соответствующие схемы, проследим, как можно выполнять обычные операции сложения в двоичной арифметике. Посмотрим еще раз на изображение двоичных чисел, соответствующих десятичным числам от 0 до 15.

Десятичные	Двоичные	Десятичные	Двоичные
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

Теперь, например, сложим числа 0001 и 0101:

```
 0101
+0001
------
    0
   1
```

В крайнем справа столбце (младший значащий разряд) 1+1=0 и перенос 1.
(сумма)
(перенос)

```
 0101
+0001
------
   10
```

Во втором справа столбце 0+0+1 (перенос)=1. Переноса в третий разряд нет.

```
 0101
+0001
------
  110
```

В третьем столбце справа 1+0=1 (переноса нет). Поэтому в сумме получается 1.

```
 0101
+0001
------
 0110
```

В крайнем левом столбце 0+0=0 (переноса нет). Поэтому в сумме получается 0.

Можно проверить этот результат по таблице соответствия двоичных и десятичных чисел: 0101 = 5 (десятичное), 0001 = 1 (десятичное).

Поскольку 5 + 1 = 6, найдем соответствующее представление десятичного числа 6 в таблице. Это 0110. Именно такой результат и получился. Попробуйте сами прибавлять двоичное число 0001 к другим двоичным числам.

Чтобы постоянно не указывать словами основание используемой системы (десятичное, двоичное и т. п.), пользуются индексами, например 16 для шестнадцатеричной системы, 10 — для десятичной, 2 — для двоичной и т. д., проставляемыми после записи соответствующего числа. Иногда вместо индекса 16 в шестнадцатеричной системе счисления используется символ H. Так, десятичное число 6 записывается как 6_{10}, двоичное 0110 — как 0110_2. Шестнадцатеричное число 0F записывается как $0F_{16}$ или как 0FH.

Выполним еще два примера на сложение. Сначала сложим числа 0110_2 и 0011_2:

$$+\begin{array}{l}0110_2\\ \underline{0011_2}\end{array}$$

Получим ответ 1001_2.

Затем перейдем к более сложному примеру: сложим числа 0111_2 и 0011_2:

$$+\begin{array}{l}0111_2\\ \underline{0011_2}\end{array}$$

С первым столбцом справа все просто: $1_2 + 1_2 = 0$ и перенос в соседний разряд 1_2. Следующий столбец кажется более сложным, но это только пока Вы не привыкли: $1_2 + 1_2$ плюс перенос из соседнего разряда справа 1_2. Известно, что $0001_2 + 0001_2 + 0001_2$ соответствует $1_{10} + 1_{10} + 1_{10} = 3_{10}$ в десятичной системе счисления, но это равно 11_2 в двоичной системе. Поступим следующим образом: в ответе под чертой пишем 1_2 и переносим 1_2 точно так же, как сделали бы при сложении двух десятичных чисел. Окончательный ответ: 1010_2.

Теперь нетрудно выполнить аналогичным образом любые действия, требующиеся при сложении двух двоичных чисел. Существует всего восемь возможных вариантов сложения двух одноразрядных двоичных чисел с учетом переносов, и, если нужно построить схему их выполнения, т. е. логическую схему, реализующую рассмотренные в примерах операции, необходимо составить таблицу истинности, в которой все восемь вариантов были бы отражены. Эти восемь вариантов, появляющиеся при

сложении одноразрядных двоичных чисел A и B с учетом переноса C, следующие:

$C=0$
$+A=0$
$B=0$
---
0 — результат или сумма

0 — переноса в следующий разряд нет

$C=1_2$ (есть перенос из предыдущего разряда)
$+A=0$
$B=0$

1_2 — сумма

0 — переноса в следующий разряд нет

$C=0$ (переноса из предыдущего разряда нет)
$+A=0$
$B=1_2$

1_2 — сумма

0 — переноса в следующий разряд нет

$C=1_2$ (есть перенос из предыдущего разряда)
$+A=0$
$B=1_2$
---
0 — сумма

1_2 — перенос в следующий разряд

$C=0$ (переноса из предыдущего разряда нет)
$+A=1_2$
$B=0$

1_2 — сумма

0 — переноса в следующий разряд нет

$C=1_2$ (есть перенос из предыдущего разряда)
$+A=1_2$
$B=0$
---
0 — сумма

1_2 — перенос в следующий разряд

$C=0$ (переноса из предыдущего разряда нет)
$+A=1_2$
$B=1_2$

---

0 — сумма

1_2 — перенос в следующий разряд

$C=1_2$ (есть перенос из предыдущего разряда)
$+A=1_2$
$B=1_2$

1_2 — сумма

1_2 — перенос в следующий разряд

Каждая такая возможность сложения в одном разряде может быть отображена в виде одной строки следующей таблицы истинности:

Входы			Выходы	
A_i	B_i	Перенос из предыдущего разряда	Сумма (S)	Перенос в следующий разряд (C)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Можно ли реализовать эту таблицу истинности схемным путем, с помощью комбинационных элементов?

Да, соответствующая схема известна под названием полный сумматор, или просто сумматор. Полный сумматор реализует операцию сложения одноразрядных двоичных чисел с учетом переноса из предыдущего разряда. Чтобы сложить два 8-разрядных слова данных, потребуется восемь полных сумматоров, параллельно соединенных друг с другом. В микро-ЭВМ два 8-разрядных числа можно сложить путем получения первого слагаемого ($A_7A_6A_5A_4A_3A_2A_1A_0$) из аккумулятора, а второго слагаемого ($B_7B_6B_5B_4B_3B_2B_1B_0$) из памяти или, возможно, из другого регистра и сложения их с помощью схемы сумма-

тора. Результат или сумма ($S_7S_6S_5S_4S_3S_2S_1S_0$) обычно затем снова поступает в аккумулятор, стирая предыдущее содержимое, после чего аккумулятор снова готов для последующего сложения и дальнейшей пересылки результата.

Полный сумматор воспринимает двоичные сигналы, поступающие на три его входа: A, B и вход переноса. Он фактически состоит из двух полусумматоров, у которых выходы переноса соединены вместе по схеме ИЛИ.

Полусумматор имеет только два входа и два выхода. Он легко может быть реализован схемным путем. Полусумматор не имеет возможности учесть сигнал переноса, так как у него всего два входа. Поэтому, чтобы получить полный сумматор, необходимо использовать схемы двух полусумматоров вместе. Именно таким образом удается сложить сигналы по входам A и B и сигнал переноса из предыдущего разряда. У такой схемы два выхода: результат или сумма S и перенос C в следующий разряд.

Таблица истинности для полусумматора имеет следующий вид:

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

т. е. строка таблицы соответствует одному из следующих вариантов суммирования:

Строка 1

$$+\begin{matrix}A=0\\B=0\end{matrix}$$

$S=0$ перенос 0

Строка 2

$$+\begin{matrix}A=0\\B=1\end{matrix}$$

$S=1$ перенос 0

Строка 3

$$+\begin{matrix}A=1\\B=0\end{matrix}$$

$S=1$ перенос 0

Строка 4

$$+\begin{matrix}A=1\\B=1\end{matrix}$$

$S=0$ перенос 1

Схемная реализация полусумматора показана на рис. 5.1 и состоит всего из двух логических элементов: ИСКЛЮЧАЮЩЕГО ИЛИ и И. Попытайтесь составить

таблицу истинности для этих элементов, чтобы убедиться, что это действительно полусумматор:

A	B	C	S
0	0		
0	1		
1	0		
1	1		

Используя таблицы истинности для логических функций И и ИСКЛЮЧАЮЩЕЕ ИЛИ, легко можно получить:

Конъюнкция (И)

A	B	Выход (C)
0	0	0
0	1	0
1	0	0
1	1	1

ИСКЛЮЧАЮЩЕЕ ИЛИ

A	B	Выход (S)
0	0	0
0	1	1
1	0	1
1	1	0

Еще раз просмотрим эти таблицы, чтобы убедиться, что все сделано правильно.

Если и это Вас не убедило, то есть еще способ проверить правильность. Сравните строки таблиц истинности полусумматора и функций ИСКЛЮЧАЮЩЕЕ ИЛИ и И путем следующих рассуждений, например для первых строк: если $A=B=0$, выходной сигнал элемента ИСКЛЮЧАЮЩЕЕ ИЛИ (являющийся выходом S) есть 0, а выходной сигнал элемента И (являющийся выходом C) есть 0.

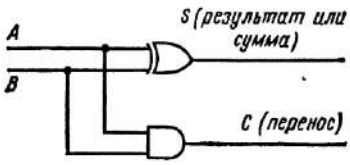


Рис. 5.1. Схема полусумматора

Далее сравниваем вторые строки каждой таблицы: если $A=0$, $B=1$, то C из таблицы И есть 0 и S из таблицы ИСКЛЮЧАЮЩЕЕ ИЛИ есть 1 и т. д.

Чтобы реализовать функцию И с помощью КМОП-структур, проще включить последовательно элементы И-НЕ и НЕ. Поэтому можно встретить схемы полусумматоров, либо выполненных, как указано на рис. 5.1, либо построенных целиком на элементах И-НЕ и инверторах.

Что такое полный сумматор?

Схема полного сумматора приведена на рис. 5.2. Она состоит из двух полусумматоров, соединенных вместе.

Для составления таблицы истинности этой схемы придется вернуться к таблицам истинности элементов ИСКЛЮЧАЮЩЕЕ ИЛИ, И и ИЛИ:

ИСКЛЮЧАЮЩЕЕ ИЛИ			И			ИЛИ		
A	B	Выход S_1 или S_2	A	B	Выход C_1 или C_2	A	B	Выход
0	0	0	0	0	0	0	0	0
0	1	1	0	1	0	0	1	1
1	0	1	1	0	0	1	0	1
1	1	0	1	1	1	1	1	1

Первая строка задается $A=0$, $B=0$, перенос из предыдущего разряда (входной перенос) $=0$. Выходной сигнал левого элемента ИСКЛЮЧАЮЩЕЕ ИЛИ S_1 равен 0 на основании приведенной таб-

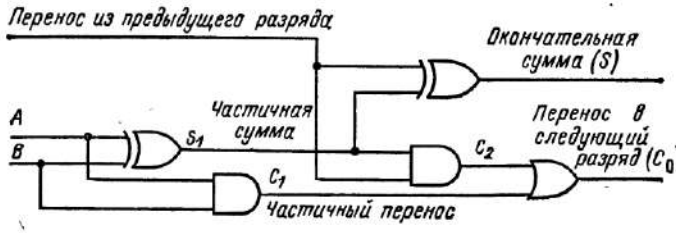


Рис. 5.2. Схема полного сумматора

лицы. Он является входным сигналом вместе с сигналом входного переноса, значение которого в данном случае равно 0, для второго элемента ИСКЛЮЧАЮЩЕЕ ИЛИ, выходной сигнал которого также равен 0 в соответствии с приведенной выше таблицей истинности. Он же является сигналом окончательной суммы S.

Сигнал на выходе C_1 левого элемента И (частичный перенос) равен 0. Выход C_1 связан со входом элемента ИЛИ; другим входом элемента ИЛИ служит выход C_2 элемента И, сигнал которого в этом случае тоже равен 0 (поскольку на обоих его входах сигналы равны 0). Поэтому сигнал выходного переноса (переноса в следующий разряд) равен 0. Проводя подобные рассуждения относительно каждой комбинации входных сигналов, можно заполнить приведенную ниже таблицу истинности для полного сумматора.

Так, например, следующая строка для входных сигналов $A=0$, $B=0$, сигнал входного переноса $=1$ будет такой: $C_1=0$, $S_1=0$, $C_2=0$, $S=1$, $C_0=0$. Предлагается вычислить все остальные значения или переписать их из таблицы восьми возможных комбинаций значений входных сигналов, имеющихся при сложении (см. выше).

Таблица истинности для полного сумматора

A	B	Сигнал входного переноса	Частичный перенос C_1	Частичная сумма S_1	Частичный перенос C_2	Окончательная сумма S	Сигнал выходного переноса C_0
0	0	0	0	0	0	0	0
0	0	1					
0	1	0					
0	1	1					
1	0	0					
1	0	1					
1	1	0					
1	1	1					

Как выполняются арифметические операции с 8-разрядными числами?

Полный сумматор выполняет сложение двух двоичных чисел с учетом переноса. Для сложения 8-разрядных чисел понадобятся восемь полных сумматоров и организация переноса путем соединения сигнала выходного переноса из самого младшего двоичного разряда с сигналом входного переноса следующего разряда и т. д. по всем восьми разрядам складываемых чисел. Перенос из самого старшего разряда может передаваться в специальное устройство микропроцессора, называемое флагом, указывающим на то, что при сложении 8-разрядных чисел существует сигнал переноса из старшего разряда. Этот сигнал необходим также при выполнении арифметических операций над 16-разрядными (и более) двоичными числами, по восемь разрядов за 1 раз. Сигнал выходного переноса из старшего разряда 8-разрядного регистра означает, что в результате операции получилось число, большее, чем $1111\ 1111_2$, которое может находиться в нем.

Что представляет собой команда сложения с переносом и без переноса?

Команда сложения с переносом (ADC) для обычного (типичного) микропроцессора предусматривает помещение любого значения сигнала выходного переноса (двоичной цифры 1) в одноразрядное запоминающее устройство, называемое флагом переноса. С учетом этого команда сложения в целом выглядит следующим образом: сложить содержимое памяти по адресу, который

будет указан сразу после кода этой операции, с содержимым аккумулятора и, кроме того, прибавить цифру входного переноса, содержащуюся в специальном запоминающем устройстве — флаге переноса; поместить затем ответ обратно в аккумулятор вместо имеющегося там числа. Если поступил сигнал окончательного выходного переноса из старшего разряда в результате этого сложения, поместить его в специальное запоминающее устройство — флаг переноса (установить флаг переноса).

Команда сложения без переноса (ADD), упоминавшаяся ранее, хотя и не предусматривает прибавление цифры входного переноса при выполнении суммирования, все-таки должна предусматривать хранение любого сигнала выходного переноса в запоминающем устройстве — флаге переноса, готового для последующего использования, если это потребуется, путем выполнения команды сложения с переносом.

Что такое сумматор с ускоренным переносом?

Простая суммирующая структура, состоящая из соединенных специальным образом полных сумматоров для выполнения операции параллельного сложения (сразу восемь разрядов), называется сумматором с последовательным переносом. Устройство не в состоянии сформулировать немедленный ответ, как можно было бы ожидать, поскольку каждый появляющийся сигнал входного переноса должен пройти поочередно последовательно через все восемь состояний. Это может вызывать значительные задержки в получении окончательного результата.

Более быстродействующий сумматор можно получить при условии, что устройство будет эффективно формировать сигналы переноса сразу для всех двоичных разрядов и сократит тем самым время задержки получения результата. Сумматор, построенный на этом принципе, носит название сумматора с ускоренным (look ahead) переносом.

Как выполняются арифметические операции над 16-разрядными числами?

Шестнадцатиразрядные числа могут складываться частями, по восемь разрядов каждая, путем суммирова-

ния сначала младших разрядов чисел (крайних восьми разрядов справа каждого числа), запоминания сигнала окончательного переноса из старшего для этой части разряда (значениями которого могут быть только 0 или 1) в одноразрядном регистре микропроцессора, называемом флагом переноса, и сложения затем этого числа (значения переноса) со следующими восемью разрядами суммируемых чисел.

Последовательность действий при сложении следующая:

1. Считать первые восемь разрядов каждого числа из памяти. Сложить эти числа, пользуясь командой ADD (СЛОЖИТЬ БЕЗ ПЕРЕНОСА). Занести результат обратно в память и переслать полученный сигнал переноса в регистр флага и переноса.

2. Считать из памяти следующие восемь разрядов каждого числа и сложить их вместе с содержимым регистра флага переноса, пользуясь командой ADC (СЛОЖИТЬ С УЧЕТОМ ПЕРЕНОСА). Занести окончательный результат в память.

Пример:

$$\begin{array}{r} 00010010 \quad 11111011_2 \\ +\,01101000 \quad 10000001_2 \\ \hline 01111011 \quad 01111100_2 \end{array}$$

был бы выполнен как сложение младших восьми разрядов

$$\begin{array}{r} 11111011_2 \\ +\,10000001_2 \\ \hline 01111100_2 \end{array}$$

с пересылкой единицы переноса в регистр флага переноса;
затем сложение старших восьми разрядов и единицы переноса

$$\begin{array}{rl} 00010010_2 & \\ +\,01101000_2 & \\ \quad\quad 1_2 & \text{(из регистра флага переноса)} \\ \hline 01111011_2 & \end{array}$$

На этот раз любой перенос будет связан с ошибкой переполнения, т. е. с получением числа, большего, чем 11111111 11111111. Переполнение уже произошло бы, если бы число 01111111 11111111 оказалось избыточным. Это объясняется тем, что крайний левый разряд обычно

используется для указания знака числа и в таком случае называется знаковым разрядом. Значение 0 при этом показывает, что данное число положительное, а 1 — что число отрицательное.

Что можно сказать о вычитании?

Рассмотрим сначала следующий пример сложения двух чисел:

$$+\begin{matrix}11111111_2\\00000001_2\end{matrix}$$

00000000_2 и перенос 1, но его пока принимать во внимание не будем.

Заметим, что 00000001_2 — это двоичная форма записи десятичного числа 1_{10}. Для получения нуля в десятичной системе счисления следует прибавить -1_{10} к $+1_{10}$:

$(-1_{10}+1_{10}=0)$.

Следовательно, двоичное число 11111111_2 можно использовать для представления десятичного отрицательного числа -1_{10}, так как при прибавлении его к $+00000001_2$ (являющемуся $+1_{10}$) получаем 00000000_2 [$A-B$ — это то же, что и $A+(-B)$].

Но если число 11111111_2 представляет собой отрицательное десятичное число -1_{10}, то как можно представить отрицательное десятичное число -2_{10}?

Очевидно, что такое число должно быть на единицу меньше числа -1_{10}. В двоичной системе счисления числом, на единицу меньшим, чем 11111111_2, будет число 11111110_2. Оно и является представлением десятичного отрицательного числа -2_{10}. Тогда представлением числа -3_{10} будет 11111101_2. Продолжая эти рассуждения, нетрудно получить следующую таблицу соответствия представлений десятичных и двоичных положительных и отрицательных чисел.

Десятичное число	Двоичное число	Десятичное число	Двоичное число
+15	0000 1111	— 1	1111 1111
+14	0000 1110	— 2	1111 1110
+13	0000 1101	— 3	1111 1101
+12	0000 1100	— 4	1111 1100
+11	0000 1011	— 5	1111 1011
+10	0000 1010	— 6	1111 1010

Продолжение

Десятичное число	Двоичное число	Десятичное число	Двоичное число
+ 9	0000 1001	— 7	1111 1001
+ 8	0000 1000	— 8	1111 1000
+ 7	0000 0111	— 9	1111 0111
+ 6	0000 0110	—10	1111 0110
+ 5	0000 0101	—11	1111 0101
+ 4	0000 0100	—12	1111 0100
+ 3	0000 0011	—13	1111 0011
+ 2	0000 0010	—14	1111 0010
+ 1	0000 0001	—15	1111 0001
0	0000 0000		

Такой способ представления положительных и отрицательных чисел принято называть представлением в дополнительном коде или в форме дополнения до 2.

При переходе к дополнительному коду представление положительных чисел не меняется, а представление отрицательных чисел можно получить путем инвертирования кода (нахождение поразрядного дополнения до 1) с последующим прибавлением единицы в младшем разряде. Представление в форме дополнения до 1 можно получить с помощью инверторов, а сложение с единицей — с помощью суммирующей схемы. Однако, как увидим далее, в организации специальных схем нет необходимости, поскольку регистр-аккумулятор может легко сформировать обратный код хранимого в нем числа.

Пусть, например, требуется получить двоичное представление десятичного отрицательного числа -1_{10} в дополнительном коде.

Положительное десятичное число 1_{10} $=00000001_2$ двоичному.
Инвертированный (обратный) код есть 11111110_2
После сложения с 1 $+\quad 1_2$

получим дополнительный код отрицательного десятичного числа -1_{10}. 11111111_2

Попробуйте сами получить аналогичное представление числа -13_{10} и проверьте результат по приведенной выше таблице.

Проделаем еще несколько вычитаний по форме $A-B=A+(-B)$.

Вычислим $5_{10}-2_{10}$ или $5_{10}+(-2_{10})=3_{10}$:
5_{10} есть 00000101_2 (уменьшаемое)
-2_{10} в дополнительном коде есть 11111110_2 (вычитаемое)

В результате сложения получим 00000011_2 (разность) и 1 переноса из старшего разряда (перенос пока оставим без внимания).

Это не является настоящим переполнением, так как остальная часть числа не есть 11111111_2. По приведенной выше таблице десятичных и двоичных чисел в дополнительном коде убеждаемся в том, что число 00000011_2 есть 3_{10}, т. е. получился правильный ответ.

Вычислим $3_{10}-5_{10}$ или $3_{10}+(-5_{10})=-2_{10}$:

3_{10} есть 00000011_2 (уменьшаемое)

-5_{10} в дополнительном коде есть 11111011_2 (вычитаемое)

В результате сложения получим 11111110_2 (разность) без переноса из старшего разряда.

По таблице убеждаемся, что полученное число 11111110_2 соответствует отрицательному десятичному числу -2_{10}, т. е. получился правильный результат.

Если эти примеры понятны, то выполните следующий. Вычислите $-3_{10}+(-5_{10})$, чтобы усвоить выполнение сложения двух отрицательных чисел $(-3_{10})+(-5_{10})=-8_{10}$:

-3_{10} в дополнительном коде есть $1111\ 1101_2$

-5_{10} в дополнительном коде есть $1111\ 1011_2$

В результате сложения получим $1111\ 1000_2$ и единицу переноса из старшего разряда.

Находим по таблице, что число 11111000_2 соответствует отрицательному десятичному числу -8_{10}; результат — правильный.

Однако как быть с возникающими в процессе вычислений переносами из крайнего левого разряда? Если проинвертировать сигнал переноса, то получится $\overline{C}$ (нет переноса). Представление $\overline{C}$ в качестве сигнала заема при выполнении вычитания может быть весьма полезным. Вычитание выполняется путем сложения уменьшаемого с вычитаемым в дополнительном коде, так что перенос при сложении в дополнительном коде служит той же цели, что и заем в обычном вычитании.

Сигнал переноса из старшего разряда 8-разрядного числа может быть использован при вычитании 16-разрядных чисел, выполняемом с участием 8-разрядного сумматора точно так же, как это делалось при сложении. Все, что необходимо при этом, это записать сигнал переноса в одноразрядный регистр флага после сложения восьми младших разрядов и учесть этот сигнал в качестве сигнала входного переноса при сложении следующих восьми разрядов.

Выполним в качестве примера вычитание следующих 16-разрядных чисел:

$01100001\ 10001001_2$ уменьшаемое
минус $01000001\ 00010001_2$ вычитаемое.

Сначала находим представление числа ($01000001\ 00010001_2$) в дополнительном коде:

обратный код вычитаемого числа	10111110	11101110_2
прибавим единицу		$+1$
получим дополнительный код	10111110	11101111_2
выполним сложение:	01100001	10001001_2
	$+10111110$	11101111_2
	00100000	01111000_2
	перенос 1	перенос 1

Полученный ответ является положительной разностью двух 16-разрядных чисел, а имеющийся перенос из крайнего левого разряда указывает на то, что знак результата совпадает со знаком уменьшаемого ($\overline{C}=0$. Заема не было, так как уменьшаемое было больше вычитаемого).

Если по окончании каждого процесса вычисления сигнал окончательного переноса отсутствует ($C=0$), то $\overline{C}=1$, т. е. должен был быть окончательный заем. Это просто означает, что вычитаемое было больше уменьшаемого.

Последний пример вычитания 16-разрядных чисел можно выполнить по частям в одном 8-разрядном сумматоре. Сначала производится сложение в дополнительном коде самых младших двоичных разрядов каждого числа:

$$\begin{array}{r} +\ 10001001_2 \\ 11101111_2 \\ \hline 01111000_2 \end{array}$$

и запоминание единицы переноса (т. е. установка флага переноса).

Этот ответ пересылается обратно в память, и выполняется следующая половина вычислений с учетом

сигнала переноса:

$$\begin{array}{r} 01100001_2 \\ +\,10111110_2 \\ \hline 1_2 \\ \hline 00100000_2 \end{array}$$

перенос из первой половины числа

и перенос 1 из второй половины числа. Эта часть результата также заносится в память

Что такое знаковый разряд?

Как видно из рассмотренных примеров, при выполнении вычислений необходим указатель знака числа (положительного или отрицательного). Для этой цели используется старший двоичный разряд (крайний слева). Его установка в состояние 0 соответствует положительному числу, а 1 — отрицательному. Этот специально выделенный разряд называется знаковым разрядом, остальные — разрядами модуля числа.

Это означает, что два 8-разрядных слова могут содержать 15 разрядов для представления модуля числа и 1 разряд для его знака. Таким образом, можно оперировать с любыми целыми числами в диапазоне от 0111111111111111_2, что соответствует положительному десятичному числу $+32767_{10}$, до 1000000000000000_2, что соответствует отрицательному десятичному числу -32768_{10}.

При выполнении операций с восьмиразрядными словами с отведенным для знака самым старшим разрядом можно оперировать с любыми целыми числами в диапазоне от 01111111_2, или $+127_{10}$, до 10000000_2, или -128_{10}.

Если попытаться сложить два больших положительных числа, превышающих в сумме 01111111_2, то 8-разрядные регистры не смогут обеспечить получение правильного результата:

$$\begin{array}{rrcl} 0 & 1111100_2 & = & 124_{10} \\ 0 & 1100111_2 & = & 103_{10} \\ \hline 1 & 1100011 & \neq & 227_{10} \end{array}$$

знак модуль

Ответ будет неверным из-за ошибки переполнения.

В действительности происходит следующее: после прибавления числа 3_{10} в регистре будет наибольшее возможное число, соответствующее заполнению единицами всех семи разрядов модуля, т. е. число 01111111_2 ($+127_{10}$). Для получения правильного результата не-

обходимо прибавить еще число $(103—3)_{10}=100_{10}$. Однако следующая единица переведет регистр в состояние 10000000, что соответствует записи числа $—128_{10}$, и останется для сложения число $(100—1)_{10}=99_{10}$. Это число в сумме с максимальным отрицательным числом регистра даст результат $(—128+99)_{10}=—29_{10}$, указанный в ответе 1100011_2 по модулю (в дополнительном коде).

Во всех случаях появления ошибки действительного переполнения, как и в данном примере, значение знакового разряда в ответе всегда будет отличаться от значений знаковых разрядов складываемых чисел, которые в случае переполнения, естественно, не отличаются друг от друга (в противном случае при сложении переполнение не произошло бы).

Аналогичную ситуацию можно наблюдать при сложении больших отрицательных чисел:

$$\begin{array}{rl} & 1\ 0000001_2 = —127_{10} \\ + & 1\ 0000011_2 = —125_{10} \\ \hline & 0\ 0000100_2 \neq —252_{10} \end{array}$$

Таким образом, можно организовать проверку условия изменения значения знакового разряда, чтобы использовать эту информацию для указания на неблагополучие при сложении путем установки в соответствующее состояние специального одноразрядного флагового регистра переполнения.

Одноразрядные флаговые регистры, такие как регистры переполнения и переноса, для удобства обычно группируются вместе, образуя в микропроцессоре специальное устройство, называемое регистром состояния или флаговым регистром.

Все арифметические и логические операции выполняются специальным устройством микропроцессора, называемым арифметическо-логическим устройством. В основе своей конструкции оно напоминает 8-разрядный сумматор. Однако существует возможность изменять выполняемые им операции, посылая в него определенные сигналы команд.

Большинство арифметическо-логических устройств микропроцессоров ориентировано на использование команды вычитания. Под действием этой команды в накапливающем сумматоре формируется обратный код данных, преобразуемый далее в дополнительный код путем прибавления единицы. Последний используется затем для сложения с содержимым указанной (определенной)

ячейки памяти, а окончательный ответ пересылается обратно в накапливающий сумматор.

Если отдельной команды вычитания, включающей в себя все эти этапы, не предусматривается, необходимо иметь несколько отдельных команд, позволяющих реализовать операцию вычитания поэтапно.

Что можно сказать об умножении и делении?

Обычно умножение и деление как отдельные операции в микропроцессорах не предусматриваются. Для их осуществления, как это будет показано позже, приходится прибегать к методу «пера и бумаги», а затем эти операции выполняются с использованием специальной подпрограммы (части программы).

Прежде чем разбираться в том, как в арифметическо-логическом устройстве производятся операции умножения и деления, действия над дробями и сверхбольшими числами, необходимо рассмотреть в деталях работу специального устройства — регистра-аккумулятора, которая так тесно связана с работой арифметическо-логического устройства в целом, что понимание этого механизма совершенно необходимо до знакомства с описанием действия всего арифметическо-логического устройства и особенностей управления им. Однако сначала придется хотя бы в общих чертах познакомиться с регистрами.

ГЛАВА ШЕСТАЯ

Регистры

Что представляют собой регистры?

Регистры — это устройства временного хранения данных, размещенные внутри микропроцессора (МП), причем некоторые из них выполняют специальные функции, другие предназначены для более общего использования. К числу последних принадлежит аккумулятор (накапливающий сумматор), тесно связанный с арифметическо-логическим устройством. В большинстве случаев обработка данных выполняется при участии аккумулятора.

Что подразумевается под комбинационной и последовательностной логикой?

Все рассматривавшиеся ранее схемы относятся к комбинационной логике. Это означает, что в каждой схеме состояние выхода зависит только от текущих состояний входов. Другими словами, значение выхода формируется немедленно вслед за изменением значений входов (если не учитывать некоторой задержки распространения сигналов). Регистры должны конструироваться таким образом, чтобы их логика была последовательностной, поскольку состояния их выходов могут зависеть от последовательности предыдущих событий (последовательности смены состояний входов). Именно эта особенность и делает их чрезвычайно полезными в качестве устройств временного хранения данных и манипулирования данными.

Как устроены регистры?

Регистры состоят из последовательно соединенных элементов, известных как триггерные схемы, или триггеры, по одному триггеру для каждого двоичного разряда регистра. Триггер — это схема, имеющая два устойчивых состояния: ВКЛЮЧЕНО, обозначаемое 1, и ВЫКЛЮЧЕНО, обозначаемое 0. Таким образом, типовая последовательность смены состояний триггера может начинаться со значений входных и выходных сигналов, равных нулю. Если значение входного сигнала равно U_+ (1), то аналогичное значение будет и на выходе. Если значение сигнала на входе изменится на первоначальное (0), то на выходе сохранится значение U_+ (1). Чтобы на выходе триггера снова установился нулевой сигнал, может потребоваться другая последовательность событий (смены входных сигналов).

Некоторые регистры допускают двунаправленную передачу данных. Некоторые, принимая данные последовательно (по одному разряду в единицу времени), осуществляют параллельную их выдачу (например, восемь двоичных разрядов в единицу времени) или наоборот. Регистры такого типа могут быть использованы в качестве портов ввода-вывода на концах линий передачи данных, в частности, для сопряжения микро-ЭВМ с внешни-

ми устройствами последовательного ввода информации, такими как накопитель на магнитной ленте, экран дисплея и т. п.

Что такое триггер, и что в нем происходит при переключении?

Проанализировать работу триггера легче всего путем замены его блоком переключателя, коммутирующего его выходную цепь с положительным полюсом источника питания U_+ землей (нулевой потенциал). Такой переключатель может занимать только два положения, показанные на рис. 6.1, *а*.

В соответствии с этим сигнал на выходе переключателя может принимать только два возможных значения: 0 или U_+, а сам переключатель оставаться в одном из двух устойчивых состояний соответственно. Поэтому он может использоваться для хранения значений двоичного числа. Значение выходного сигнала U_+ соответствует двоичной 1 в запоминающем устройстве, а нулевое значение выходного сигнала — двоичному 0.

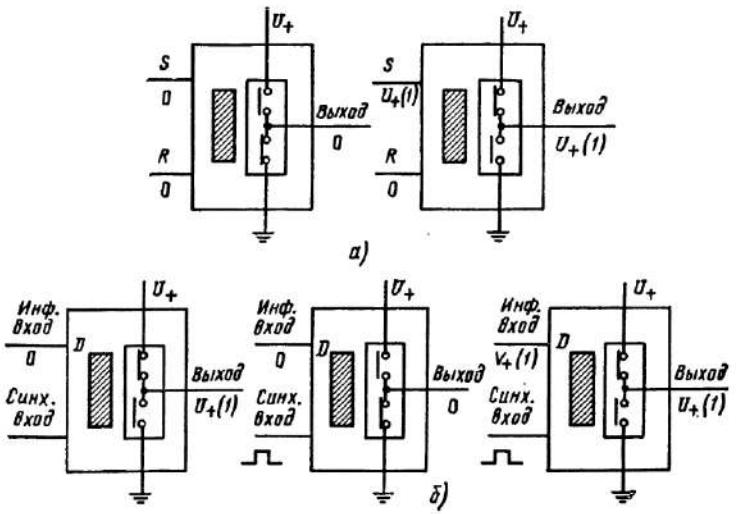


Рис. 6.1. Триггер на базе переключателя:

а — *RS*-триггер; *б* — *D*-триггер с синхронизирующим входом для различных комбинаций входных сигналов

Нетрудно заметить, что в действительности выход может быть реализован с применением типовой схемы инвертора в совокупности с некоторой другой схемой, позволяющей блокировать (фиксировать) значение выходного сигнала в каждом из устойчивых состояний (см. заштрихованную область на рис. 6.1). Далее вернемся к этой схеме, но, прежде чем сможем в ней разобраться, необходимо рассмотреть различные типы входов схемы триггера.

Рассмотрим еще раз триггер на рис. 6.1, *а* со входами S и R. Предположим, что значение его выходного сигнала неизвестно (либо 0, либо 1). Вход R (reset — очистка) предназначен для перевода триггера в состояние 0, или, как говорят, для очистки триггера. Когда он соединен с U_+, на выходе триггера появляется нулевой сигнал, если до этого момента был сигнал U_+, и сохраняется нулевой сигнал в противном случае. Триггер остается в нулевом состоянии, даже если в дальнейшем значение входа R изменится на 0. Последующее соединение R с U_+ не приведет к изменению состояния триггера до тех пор, пока он снова не окажется в состоянии 1. Это может быть выполнено с помощью другого входа S (set—установка). Появление потенциала U_+ на этом входе заставит переключатель снова изменить значение выходного сигнала на U_+. Триггер остается в этом устойчивом состоянии независимо от того, что происходит затем со входом S, до тех пор, пока на входе R не появится опять потенциал U_+, вследствие чего на выходе триггера снова будет сигнал 0.

Этот триггер называется триггером типа RS, или RS-триггером.

Существуют также синхронизируемые (тактируемые) RS-триггеры, для переключения которых необходима подача тактового импульса одновременно с поступлением сигналов на входы S или R.

Триггеры RS-типа работают устойчиво только при условии, что на их входах имеются противоположные сигналы; если на входы S и R одновременно поступит положительный потенциал U_+, значение выходного сигнала непредсказуемо[1].

---

[1] Заметим, что значения $S=R=0$ допустимы. При этом триггер сохраняет свое предыдущее состояние. Этот факт следует из приведенного автором описания работы триггера (см. также левый триггер на рис. 6.1, *а*). *(Прим. ред.)*

Этот недостаток отсутствует в триггерах типа D, или в D-триггерах. Такой тип триггера имеет только один информационный вход D (data — данные), который либо устанавливает триггер в состояние 1 (при значении входного сигнала U_+), либо производит сброс триггера в состояние 0 (при нулевом входном сигнале). Синхронизируемый D-триггер имеет, кроме входа D, еще один так называемый отпирающий, или синхронизирующий, вход C (clock). Входные сигналы (U_+ или 0) передаются к выходу триггера только в том случае, если на синхронизирующем входе имеется сигнал U_+ (рис. 6.1, *б*).

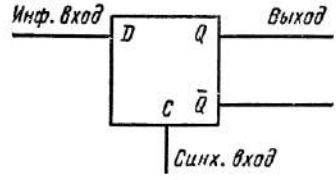


Рис. 6.2. D-триггер

Обычная последовательность событий (смены сигналов) может выглядеть следующим образом:

1. На выходе D-триггера уже имеется сигнал U_+.
2. Нулевой потенциал на входе D при появлении тактового импульса U_+ сбросит триггер в 0 (сигнал на выходе будет равен 0), а потенциал U_+ на входе D при появлении тактового импульса оставит триггер в состоянии 1 (сигнал на выходе будет равен U_+).

Смена сигналов на входе D при отсутствии тактового импульса не оказывает влияния на выходной сигнал триггера. Схемное изображение D-триггера приведено на рис. 6.2.

Каким образом запоминаются двоичные числа?

Совместное использование восьми триггеров позволяет запоминать 8-разрядные двоичные числа. В действительности триггер можно рассматривать как двойной переключатель, поэтому практически у него имеются два выхода, сигнал на одном из которых $\bar{Q}$ всегда противоположен по значению сигналу на другом выходе Q.

Как загружаются и разгружаются регистры?

И то и другое можно выполнить параллельно, по восемь двоичных разрядов за один временно́й интервал, или последовательно, по одному разряду в единицу времени.

Параллельный способ должен быть вполне очевиден. Если каждая линия шины данных соединена с информационным входом отдельного *D*-триггера, то, как только на их синхронизирующих входах появится потенциал U_+, они запомнят информацию, имеющуюся на информационной шине, и будут продолжать ее хранить до тех пор, пока не поступит следующий тактовый импульс, при котором их выходные сигналы изменятся в соответствии с новыми данными, имеющимися в линиях передачи данных.

При последовательном вводе регистр в основном остается тем же, однако выход каждого триггера должен быть соединен со входом следующего. При этом данные поступают поразрядно на вход первого триггера синхронно с тактовыми импульсами, продвигаются поразрядно на вход следующего триггера и т. д. пока не заполнят весь регистр.

Это не совсем так просто, как кажется, поскольку если бы этим ограничивалось все здесь описанное, можно было бы передавать каждый двоичный разряд сразу через все ступени, и запись, очевидно, закончилась бы последовательностью сигналов 0000 0000 или 1111 1111 в запоминающем устройстве в зависимости от значения последнего введенного двоичного разряда.

Для преодоления подобного «domino effect»[1] необходимо воспользоваться специальной схемой триггера, известного как триггер, переключаемый по срезу входного импульса, либо схемой триггера типа *MS* (Master — Slave), или *MS*-триггера[2]. С их помощью можно также избавиться от проблем согласования переключений во времени, которые могут возникнуть, например, если время переключения триггера меньше длительности тактового импульса.

Представьте себе *MS*-триггер как два триггера, последовательно включенных вместе, в одном блоке. Они организованы так, что при поступлении тактового (синхронизирующего) импульса первый триггер устанавливается в 1, а второй блокируется. В соответствии с этим на выход первого триггера поступает тот же сигнал, который поступил ранее на его информационный вход.

---

[1] Эффекта завершения. *(Прим. пер.)*

[2] Двухступенчатый триггер, или триггер, построенный по принципу двухступенчатого запоминания информации. *(Прим. пер.)*

Когда тактовый импульс исчезает, второй триггер устанавливается в 1, а первый блокируется. Поэтому выходной сигнал первого триггера передается на вход второго. Таким путем в схему вводится задержка и устраняется «domino effect».

Существует еще одна модель триггера, а именно JK-триггер. У него два выхода Q и $\bar{Q}$ и два входа J и K, и

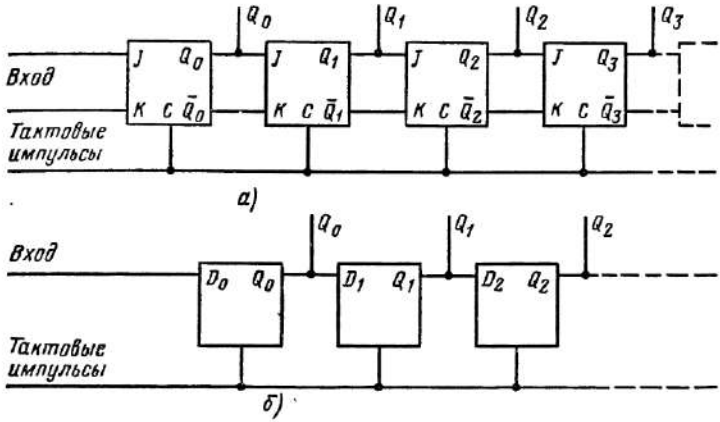


Рис. 6.3. Сдвиговые регистры с последовательным вводом данных: *а* — на базе JK-триггеров; *б* — на базе MS-триггеров типа D

возможен также синхронизирующий вход. Он аналогичен RS-триггеру, однако для него допустимы значения входных сигналов $J=K=1$ (вход J соответствует входу S, а вход K — входу R).

Триггеры типа MS на базе JK-триггеров могут соединяться вместе, образуя регистр с последовательным вводом данных (рис. 6.3, *а*).

Для первой ступени регистра с последовательным вводом часто организуется единственный информационный вход, иногда называемый входом JK. Это выполняется путем соединения входа K со входом J через логический элемент НЕ, реализующий функцию отрицания.

Триггеры MS на базе D-триггеров также можно использовать для построения регистра с последовательным вводом данных (рис. 6.3, *б*).

Помимо уже упомянутых входов, у многих регистров имеются также входы общего обнуления, которые уста-

навливают каждый триггер в нулевое состояние (с нулевым выходным сигналом) независимо от наличия тактовых импульсов. Иногда предусматриваются также входы общей загрузки, устанавливающие каждый триггер в состояние 1.

В некоторых регистрах предусматривается управление сдвигом и загрузкой. Это позволяет загружать регистр данными параллельно, а затем считывать их последовательно (путем сдвига), и наоборот.

Как построить триггеры из рассмотренных ранее базовых схем инверторных переключателей?

Рассмотрим сначала простой *RS*-триггер. Представим себе два последовательно включенных инвертора, охваченных петлей обратной связи; первый имеет вход S и выход $\bar{Q}$, а второй — вход R и выход Q. На рис. 6.4, *а* приведена соответствующая схема в символах рассмотренных ранее переключателей, а на рис. 6.4, *б* — схема из инверторов.

Если сигнал на выходе Q равен 0, то сигнал на входе первого инверторного переключателя также равен 0; поэтому на его выходе сигнал будет равен U_+, и он же, будучи входным сигналом второго переключателя, под-

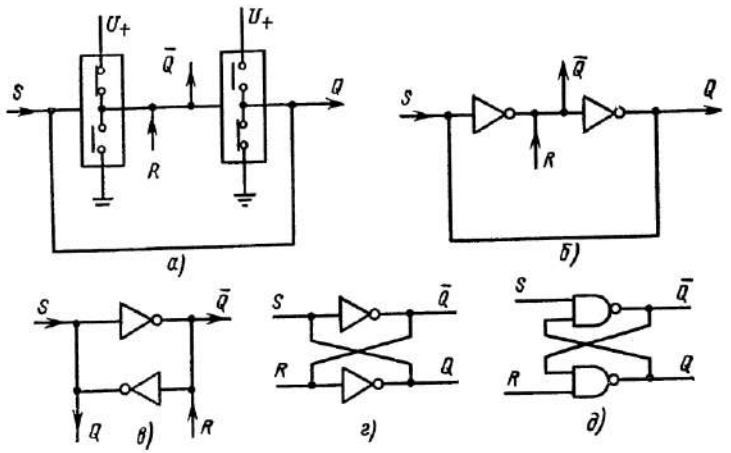


Рис. 6.4. *RS*-триггер:

а — модель на базе переключателей; *б* — модель на базе двух инверторов; *в* — модель из двух инверторов, изображенная иначе; *г* — модель из двух инверторов с разделением входов и выходов; *д* — схема на элементах И-НЕ

держивает нулевое значение на его выходе. Это условие устойчивости может быть записано в следующем виде:

$S=0;\ Q=0;$
$R=U_+;\ \overline{Q}=U_+.$

Если на входе S появится сигнал U_+ (рис. 6.4, *б*), первый инвертор переключится, и на его выходе сигнал будет равен 0. Поскольку сигнал на входе второго инвертора стал нулевым, последний тоже переключится, и сигнал на его выходе изменится на U_+. Будучи поданным по цепи обратной связи на вход S, сигнал U_+ поддержит всю схему в устойчивом состоянии даже в том случае, когда внешний сигнал U_+, поданный ранее на вход S, исчезнет. Таким образом, рассматриваемая схема может быть установлена в устойчивое состояние с сигналом на выходе Q, равным U_+ (состояние 1), путем подачи сигнала U_+ на вход S. Аналогичным образом схема может быть переведена в устойчивое состояние 0 путем подачи сигнала U_+ на вход R. Следовательно, это и есть RS-триггер.

Точно так же, как были составлены таблицы истинности для функций комбинационной логики, можно составить аналогичные таблицы, называемые иногда операционными или таблицами действий[1], для последовательностной логики рассматриваемых триггеров. Единственное отличие состоит в том, что, поскольку логика последовательностная, значение следующего выходного сигнала зависит от текущих значений как входных сигналов, так и выходного. Поэтому в таблице переходов указываются текущие значения Q, S и R, чтобы предсказать следующее значение выходного сигнала Q:

Текущее значение Q	Текущее значение S	Текущее значение R	Следующее значение Q	Примечание
0	0	0	0	Устойчивые состояния не меняются
1	0	0	1	
0	1	0	1	Сигнал на S переводит Q в 1
1	1	0	1	

[1] Более распространенное название для таких таблиц устройств последовательностной логики — таблицы переходов. *(Прим. ред.)*

Продолжение

Текущее значение Q	Текущее значение S	Текущее значение R	Следующее значение Q	Примечание
0 0	0 0	1 1	0 0	Сигнал на R переводит Q в 0
0 1	1 1	1 1	Непредсказуемо	Значение сигналов на входах недопустимы

В данной таблице 1 соответствует значению сигнала U_+, а 0 — нулевому потенциалу. Заметим, что при $S=R=1$ следующее значение выхода непредсказуемо, так как наличие таких входных сигналов может привести как к устойчивому состоянию $Q=1$, $\overline{Q}=0$, так и к устойчивому состоянию $Q=0$, $\overline{Q}=1$. На практике такой случай часто исключается путем организации единственного входа с помощью элемента НЕ, включенного между входами S и R подобно тому, как это делалось в D-триггерах.

В ряде случаев может понадобиться триггер, работающий так же, как RS-триггер, но допускающий наличие значений входных сигналов $R=S=1$. Таким и является JK-триггер. При $R=S=1$ он перебрасывается в другое состояние.

Если входы JK-триггера соединить вместе, то получится триггер с двумя устойчивыми состояниями, меняющий свое состояние каждый раз, когда на его входе появляется сигнал U_+. Этот вид триггеров, кроме информационного входа, может иметь также синхронизирующий вход. Он называется T-триггером[1].

Как организовать связи между триггерами в регистре таким образом, чтобы можно было записывать и считывать информацию в последовательной и параллельной форме?

Это достигается путем включения логических элементов между триггерами регистра. С помощью логических элементов устанавливаются различные пути для

[1] А также-триггером со счетным входом, или счетным триггером. *(Прим. ред.)*

прохождения данных в зависимости от того, какие на них подаются управляющие сигналы.

Чтобы организовать различные пути для прохождения данных, можно воспользоваться логической операцией И. Элемент И может открываться подачей управляю-

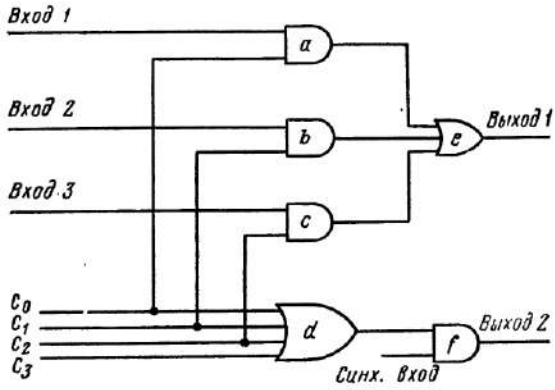


Рис. 6.5. Схема управления

щего сигнала на один из его входов и закрываться при исчезновении управляющего сигнала.

Например, в схеме, изображенной на рис. 6.5, в зависимости от значений управляющих сигналов C_0—C_3 сигналы со входов *1*—*3* либо проходят на выход *1*, либо нет, и в то же время сигналы синхрогенератора либо проходят на выход *2*, либо нет.

Посмотрим, как работает эта схема. Если $C_0=C_1=C_2=C_3=0$, то сигнал на выходе элемента *d* будет равен 0. Следовательно, будет равен 0 сигнал на одном из входов элемента *f*, и сигнал на его выходе будет равен 0 независимо от наличия или отсутствия тактовых импульсов. Если 1 появляется на любом из входов C_0, C_1, C_2 или C_3, то тактовые импульсы будут проходить через элемент *f* на выход *2*. При $C_0=C_1=C_2=C_3=0$ на одном из входов элементов *a*, *b*, *c* также будут нулевые сигналы, поэтому сигналы на их выходах равны 0 так же, как и на выходе *1* элемента *e*.

Пусть управляющие сигналы составляют набор значений 0001, т. е. $C_0=1$, $C_1=C_2=C_3=0$. В этом случае сигнал на выходе элемента *a* будет аналогичен сигналу на входе *1*, который через элемент *e* поступит на выход *1*. Элементы *b* и *c* все еще будут иметь по нулевому сигналу на одном из входов, поэтому их выходные сигналы равны 0.

Точно так же можно убедиться в том, что при наборе управляющих сигналов 0010, т. е. когда $C_0=0$, $C_1=1$, $C_2=C_3=0$, вход *2* будет соединен с выходом *1*, а вход тактовых импульсов — с выходом *2*.

При наборе управляющих сигналов 0100 вход *3* соединяется с выходом *1*, а вход тактовых импульсов — с выходом *2*.

При наборе управляющих сигналов 1000 на выходе *1* будет нулевой сигнал, а тактовые импульсы будут проходить на выход *2*.

Если управляющие сигналы запоминаются с помощью триггеров, то вместе с сигналами C_0—C_3 будут сформированы и их инверсные сигналы $\overline{C}_0$—$\overline{C}_3$, и те же самые логические функции могут быть реализованы более простым способом.

Рассмотрим особенности выполнения логических операций И и ИЛИ. Это приведет к лучшему пониманию логики функционирования практических схем.

Проанализируем, как с помощью логических схем и связей между триггерами с единственным входом типа *MS* можно построить регистр, в который можно записывать и из которого можно считывать данные в последовательной и параллельной форме. На рис. 6.6 приведена схема 8-разрядного регистра, причем для простоты показаны только четыре триггера: Q_0, Q_1, Q_2 и Q_7.

При наборе управляющих сигналов 0001 (рис. 6.7) схема функционирует как сдвиговый регистр, и двоичные разряды записываемых данных поступают по одному в каждый такт времени на левый последовательный вход схемы синхронно с импульсами синхрогенератора и сдвигаются далее вдоль регистра вправо (жирные линии на рис. 6.7). Данные могут считываться либо параллельно с выходов Q_0, Q_1, ..., Q_7, либо последовательно с правого выхода схемы (с выхода триггера Q_7).

При наборе управляющих сигналов 0010 (рис. 6.8) вход каждого триггера оказывается соединенным с выходом предыдущего справа, а двоичные разряды записываемых данных поступают на правый последовательный вход схемы синхронно с импульсами синхрогенератора и сдвигаются вдоль регистра влево. Данные могут считываться либо с выходов Q_0, Q_1, ..., Q_7 параллельно, либо последовательно с левого последовательного выхода схемы, соединенного с выходом триггера Q_0.

При наборе управляющих сигналов 0100 (рис. 6.9) данные поступают на входы каждого из триггеров в параллельной форме (на параллельные входы схемы регистра) в течение времени, равного длительности одного импульса синхронизации, и могут считываться параллельно с соответствующих выходов.

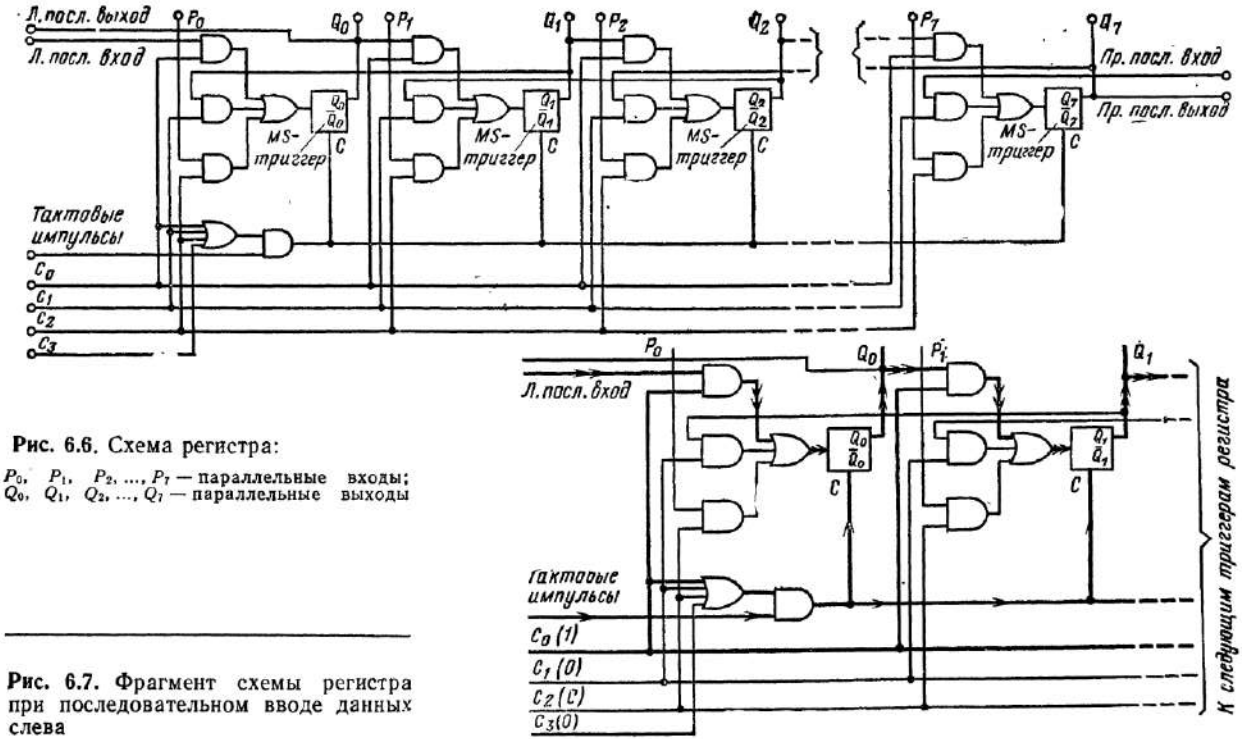


Рис. 6.6. Схема регистра:

P_0, P_1, P_2, ..., P_7 — параллельные входы;
Q_0, Q_1, Q_2, ..., Q_7 — параллельные выходы

Рис. 6.7. Фрагмент схемы регистра при последовательном вводе данных слева

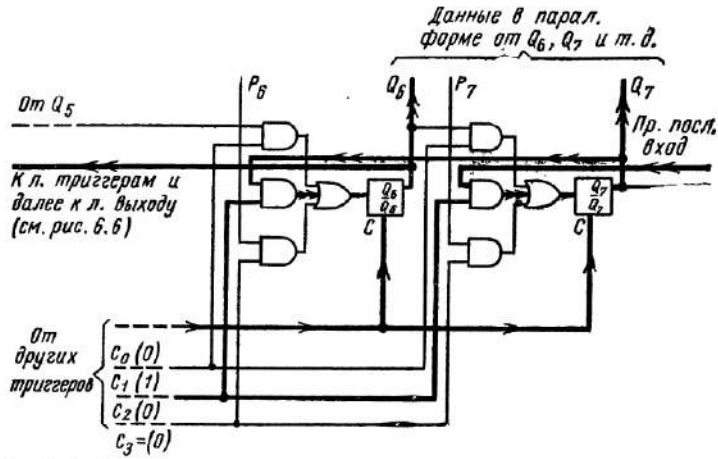


Рис. 6.8. Фрагмент схемы регистра при последовательном вводе данных справа

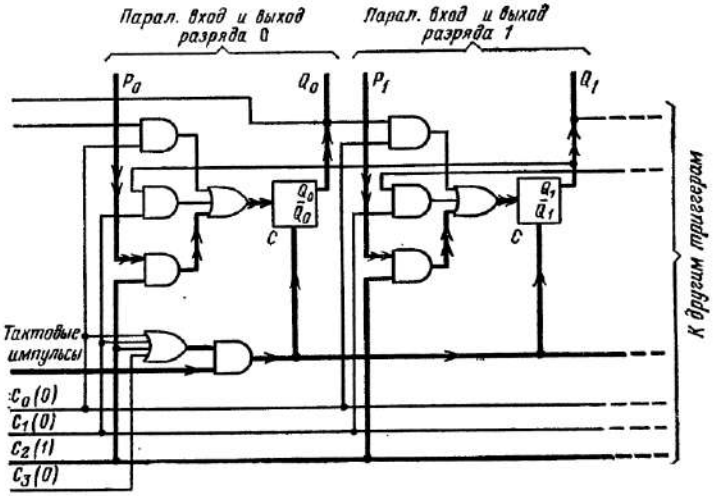


Рис. 6.9. Левый фрагмент схемы регистра при параллельном вводе и выводе данных

При наборе управляющих сигналов 1000 на входах всех триггеров будет нулевой сигнал, поэтому при поступлении следующего синхроимпульса на выходах всех триггеров установятся нулевые сигналы (произойдет очистка регистра).

Обладает ли регистр-аккумулятор возможностью сдвига данных влево и вправо?

Да, в аккумуляторе сдвиг данных используется в целях облегчения выполнения некоторых операций арифметическо-логическим устройством.

Как регистр-аккумулятор связан с арифметическо-логическим устройством?

Регистр-аккумулятор связан с арифметическо-логическим устройством через логические элементы, с помощью которых осуществляется управление его входами и выходами, так же, как управляются входы и выходы сдвигающих регистров. Многие арифметические и логические операции могут выполняться организацией различных путей прохождения данных и пересылки их к соответствующим входам в схемах сдвигающих регистров. Наиболее общими являются такие операции, как СЛОЖЕНИЕ (ADD), ПРИРАЩЕНИЕ НА 1 (INCREMENT BY 1), ОЧИСТКА, или СБРОС (CLEAR), И (AND), ИЛИ (OR), ИСКЛЮЧАЮЩЕЕ ИЛИ, или СЛОЖЕНИЕ ПО МОДУЛЮ 2 (XOR), СДВИГ ВПРАВО (SHIFT RIGHT), СДВИГ ВЛЕВО (SHIFT LEFT), ВЗЯТИЕ ДОПОЛНЕНИЯ ДО 1 (COMPLEMENT), СРАВНЕНИЕ (COMPARE), ПРОВЕРКА НА ОТРИЦАТЕЛЬНОСТЬ (NEGATIVE CHECK), ПРОВЕРКА НА НУЛЬ (ZERO CHECK). Точный перечень предусмотренных операций варьируется в зависимости от типа микропроцессора.

Различные входы схем, необходимые для выполнения различных операций, связаны со входом каждого триггера через элемент ИЛИ. Подробное рассмотрение всевозможных схем было бы утомительно. Разобраться в том, как работает арифметическо-логическое устройство, можно, проанализировав цепи, связывающие его с аккумулятором, необходимые для обеспечения выполнения следующей простой программы: ОЧИСТКА (CLEAR), ЗАГРУЗКА (LOAD), СЛОЖЕНИЕ (ADD) и ХРАНЕНИЕ (STORE).

Последовательность операций в типовом микропроцессоре могла бы быть следующей: очистить флаг переноса, записать (запомнить) число ($A_7A_6A_5A_4A_3A_2A_1A_0$) в аккумулятор, запомнить другое число ($B_7B_6B_5B_4B_3B_2B_1B_0$) в буферном регистре, сложить их в сумматоре и по-

местить сумму снова в аккумулятор вместо первого числа. Далее можно произвести последующие операции с суммой, такие как приращение ее на 1, инвертирование или пересылка на хранение в память.

За исключением сквозной передачи сигналов переноса, сложение производится параллельно. Поэтому можно упростить анализ работы схемы, не рассматривая цепи

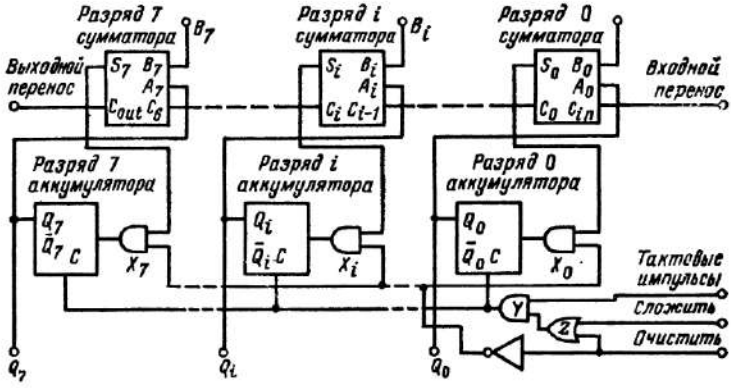


Рис. 6.10. Самый старший, i-й и самый младший двоичные разряды регистра-аккумулятора

сдвига, которые уже исследовались, и концентрируя внимание на одном двоичном разряде (разряд i) регистра, находящемся где-нибудь в середине. Он показан в схеме регистра-аккумулятора на рис. 6.10. Там же приведены схемы первого и последнего разрядов, чтобы можно было видеть, что происходит с сигналом входного и выходного переноса для всего регистра. Принято называть крайний правый разряд регистра (самый младший двоичный разряд) нулевым, крайний левый разряд (самый старший двоичный разряд) седьмым.

Предположим, что код входных управляющих сигналов (СЛОЖИТЬ и ОЧИСТИТЬ) есть 00, а одно из складываемых чисел поступило в регистр-аккумулятор и хранится в нем, о чем можно судить по сигналам на его выходах $Q_7Q_6Q_5Q_4Q_3Q_2Q_1Q_0$. Другое число поступило в буферный регистр, и соответствующие сигналы имеются на его выходах $B_7B_6B_5B_4B_3B_2B_1B_0$. Схема сумматора (верхние блоки на рис. 6.10) комбинационная, поэтому на его выходах ($S_7S_6S_5S_4S_3S_2S_1S_0$) уже есть сигналы

суммы, а на выходах ($C_7C_6C_5C_4C_3C_2C_1C_0$) сигналы переносов.

Результат суммирования подается обратно на входы регистра-аккумулятора через элементы И $X_7X_6X_5X_4X_3X_2X_1X_0$. Это возможно благодаря тому, что на других входах этих элементов присутствуют единичные сигналы, поскольку они соединены через элемент НЕ с управляющим сигналом ОЧИСТИТЬ, а его код есть 0. Результат суммирования пока не влияет на содержимое регистра-аккумулятора, так как схема его не комбинационная, а последовательностная, и в этот момент импульс синхронизатора не проходит через элемент *Y*. Это связано с тем, что как управляющий сигнал СЛОЖИТЬ, так и сигнал ОЧИСТИТЬ имеют нулевые коды, которые, будучи поданными на схему ИЛИ (элемент *Z*), обнуляют нижний вход элемента *Y* (схемы И) и отключают тем самым цепь синхронизации. Если на управляющий вход СЛОЖИТЬ поступит сигнал 1 в течение времени прохождения очередного тактового импульса, то этот импульс поступит на все синхронизирующие входы аккумулятора через элемент *Y* (схему И). Это приведет к тому, что сигналы результата сложения станут входными сигналами первой ступени каждого *MS*-триггера. После прохождения тактового импульса сумма автоматически перепишется из первой ступени каждого *MS*-триггера во вторую и появится на выходах, где и будет храниться, если на управляющих входах снова будут сигналы 00. Сумму теперь можно считать в параллельной форме с выходов аккумулятора $Q_7Q_6Q_5Q_4Q_3Q_2Q_1Q_0$ или в последовательной форме с использованием описанных ранее цепей.

Если потребуется выполнить команду ОЧИСТИТЬ аккумулятор, то это можно сделать подачей управляющих сигналов 01 в течение длительности тактового импульса. Это приведет к обнулению выходов схем И (элементов $X_7X_6X_5X_4X_3X_2X_1X_0$), и, следовательно, к тому, что на всех входах аккумулятора в течение времени, равного длительности импульса синхронизации, будут нулевые сигналы. К моменту, когда тактовый импульс прекратится, на выходы всех триггеров поступят 0. Если выходной сигнал последнего переноса аккумулятора на выходе разряда 7 поступит на триггер флага переноса, то этот триггер будет установлен в состояние 0 после выполнения указанной выше операции.

На практике часто предусматривается отдельная команда ОЧИСТИТЬ ФЛАГ ПЕРЕНОСА (CLEAR CARRY FLAG), используемая для установки триггера флага переноса в состояние 0 без изменения содержимого накапливающего сумматора.

Если потребуется увеличить содержимое аккумулятора на 1, то необходимо воспользоваться схемой, запре-

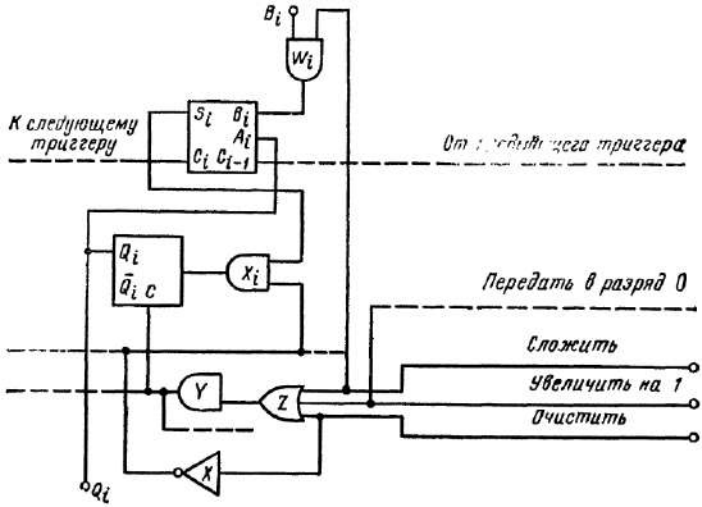


Рис. 6.11. Схема сумматора с элементами W_i (схемы И), подключающим входы B_i только при наличии сигнала управления СЛОЖИТЬ

щающей передачу любого числа в буферный регистр (либо произвести сначала его очистку). Затем организуем передачу числа, хранящегося в аккумуляторе, в сумматор и передадим 1 в качестве начального переноса в C_0. Одновременно дадим возможность тактовому импульсу пройти ко всем триггерам аккумулятора. Единица переноса, естественно, сложится с числом, и в аккумуляторе будет содержаться требуемая сумма, т. е. исходное число, увеличенное на 1. Оно появиться в аккумуляторе после прохождения тактового импульса.

На рис. 6.11 представлена логика организации схемного решения (для i-го разряда).

Элементы $W_7 \ldots W_i \ldots W_0$ (схемы И) позволяют содержимому буферного регистра перейти в сумматор только тогда, когда на входе

СЛОЖИТЬ имеется сигнал 1. Когда на управляющих входах УВЕЛИЧИТЬ НА 1, СЛОЖИТЬ, ОЧИСТИТЬ имеется соответствующий набор сигналов 010, число в буферном регистре прибавляется к содержимому аккумулятора. Если набор управляющих сигналов есть 100, то содержимое аккумулятора и перенос единицы младшего разряда являются входными сигналами для сумматора. Если набор управляющих сигналов есть 001, то производится, как и ранее, очистка аккумулятора.

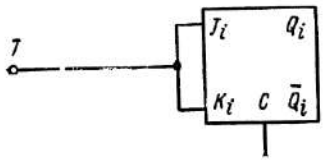


Рис. 6.12. *JK*-триггер в качестве *T*-триггера

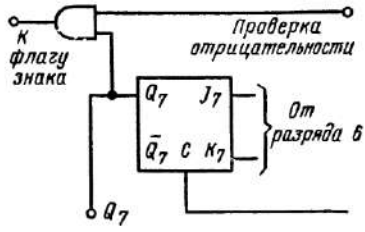


Рис. 6.13. Схема проверки на отрицательность

Для простоты рассматривались только схемы с одновходовыми триггерами. На практике обычно используются триггеры с двумя входами типа *JK*, которые путем включения дополнительных логических элементов легко могут быть превращены по желанию в *MS*-триггеры типов $J\bar{K}$, *D* или *T*. Представляет, например, интерес построение аккумулятора из *T*-триггеров и исключение всех других входов в него, так как, когда сигнал U_+ (1) подается на вход каждого триггера, содержимое аккумулятора инвертируется при поступлении следующего тактового импульса. Это полезно для выполнения операции нахождения дополнения двоичного числа путем прибавления к инвертированному значению кода числа единицы в целях получения дополнения до двух для последующего вычитания (рис. 6.12).

Если требуется выяснить, отрицательное ли число находится в аккумуляторе, то необходимо выход аккумулятора Q_7 (значение сигнала на котором 1 для отрицательного числа и 0 для положительного) соединить по схеме И с управляющим входом схемы проверки на отрицательность, имеющим значение 1. Если входной сигнал окажется равным 1, то этот сигнал можно использовать для установки одноразрядного указателя отрицательности числа — флагового регистра (обычно называемого флагом или разрядом знака) (рис. 6.13).

Проверка на нуль выполняется аналогично. Все инверсные выходы аккумулятора $\bar{Q}_0\bar{Q}_1\bar{Q}_2\bar{Q}_3\bar{Q}_4\bar{Q}_5\bar{Q}_6\bar{Q}_7$ соединяются с входом проверки на 0 по схеме И. На выходе элемента И будет сформирован единичный сигнал только в том случае, когда все триггеры аккумулятора находятся в состоянии 0, т. е. значения сигналов на инверсных выходах равны 1. Этот сигнал может быть использован для установки флага нуля (рис. 6.14).

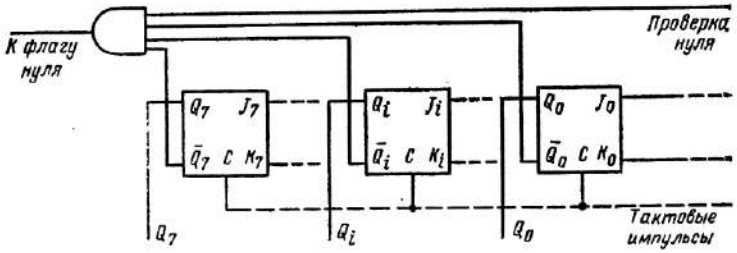


Рис. 6.14. Схема проверки на нуль

С помощью регистра-аккумулятора можно реализовать логические операции. Например, если требуется выполнить ЛОГИЧЕСКОЕ УМНОЖЕНИЕ (LOGIC AND) числа, находящегося в буферном регистре, и числа, находящегося в аккумуляторе, то это можно сделать подачей инвертированных сигналов *B* на входы аккумулятора *K*. ЛОГИЧЕСКОЕ СЛОЖЕНИЕ (LOGIC OR) может выполняться подачей сигналов *B* на входы аккумулятора *J*, а ИСКЛЮЧАЮЩЕЕ ИЛИ (LOGIC EXCLUSIVE OR)— подачей сигналов *B* на входы *J* и *K* одновременно. Если этот результат затем инвертируется, то это значит, что выполнена операция ЭКВИВАЛЕНТНОСТИ (LOGIC COINCIDENCE), т. е. произведено сравнение двух чисел, чтобы выяснить, равны ли они.

Итак, читателю ясно, как арифметическо-логическое устройство и его аккумулятор могут СКЛАДЫВАТЬ, производить ПРИРАЩЕНИЕ НА 1, ОЧИЩАТЬ, СДВИГАТЬ ВЛЕВО, СДВИГАТЬ ВПРАВО, получать ДОПОЛНЕНИЕ, СРАВНИВАТЬ, выполнять логические операции И, ИЛИ, ИСКЛЮЧАЮЩЕЕ ИЛИ и ПРОВЕРКУ НА ОТРИЦАТЕЛЬНОСТЬ числа и ПРОВЕРКУ НА НУЛЬ. Эти операции дают возможность также реализовать умножение и деление, а также арифметические операции с плавающей запятой (см. гл. 7).

Умножение и деление

Как выполняются умножение и деление арифметическо-логическим устройством?

Для умножения и деления обычно не предусматривается отдельных специальных команд, а если они есть, то выполняются арифметическо-логическим устройством (АЛУ) точно так же, как это делается обычным способом (карандашом на бумаге).

Рассмотрим пример умножения двух двоичных чисел:

```
 ×00001010   множимое
  00001011   множитель
 ----------
```

Сначала множимое умножается на цифру самого младшего двоичного (правого) разряда множителя:

```
 ×00001010   (чтобы получить ответ, умножим 0000 1010 на 1)
  00001011
 ----------
  00001010   = первое частичное произведение
```

Затем умножаем множимое на цифру во втором разряде множителя справа и смещаем второе частичное произведение влево на один разряд (так же, как и в десятичной арифметике):

```
  00001011
 ----------
 00001010
  0001010   = второе частичное произведение
```

Повторяем это с каждой из цифр разрядов множителя по очереди и затем складываем полученные частичные произведения:

```
 ×00001010
  00001011
 ----------
  00001010 }
   0001010 }
 +  000000 } частичные произведения
   01010   }
 ----------
  01101110   = окончательное произведение
```

Окончательное произведение должно быть помещено в регистр и по значению должно быть меньше числа 01111111, если оно положительное.

Описанная последовательность процедур становится выполнимой в микропроцессоре при условии размещения множимого и множителя в отдельных сдвиговых регистрах. Выходы сдвигового регистра множимого соединены со входами аккумулятора. Самый младший (пра-

вый) двоичный разряд регистра множителя соединен со схемой, которая может, когда потребуется, подключить его к схеме управления сложением арифметическо-логического устройства. На входах B в арифметическо-логическом устройстве сигналы равны 0 (рис. 7.1).

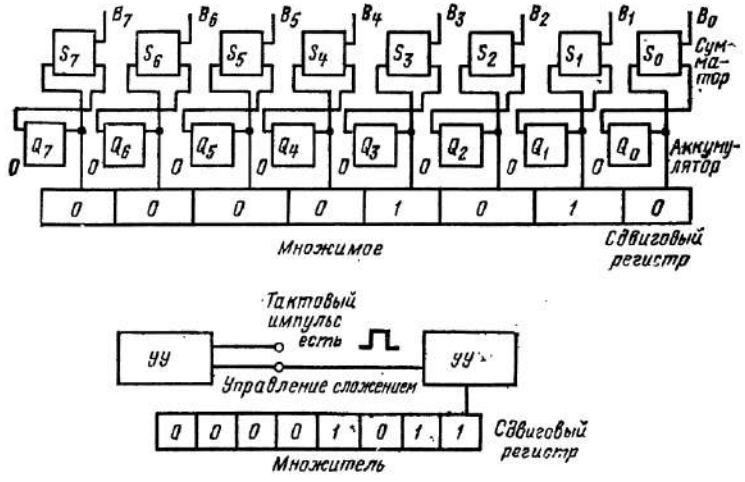


Рис. 7.1. Структурная схема умножения:
УУ — устройство управления

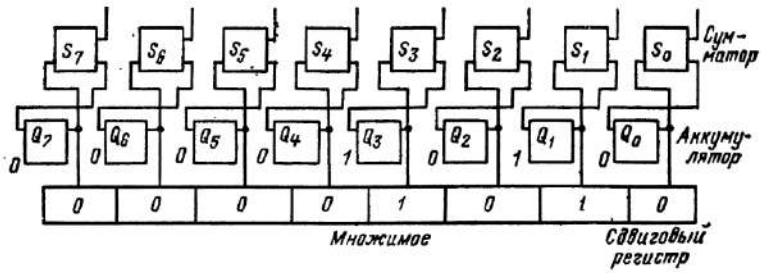


Рис. 7.2. Образование первого частичного произведения

Во время поступления очередного тактового импульса правый двоичный разряд множителя подключается к схеме управления сложением АЛУ. Если цифрой правого разряда множителя является 1, то множимое поступает в аккумулятор, чтобы образовать первое частичное произведение, и его соответствующие сигналы появляются на выходах $Q_7Q_6Q_5Q_4Q_3Q_2Q_1Q_0$ к моменту окончания тактового импульса, что и показано на рис. 7.2.

Далее содержимое сдвигового регистра множимого сдвигается на один разряд влево, а содержимое сдвигового регистра множителя — на один разряд вправо. Во время этих операций очередной тактовый импульс в аккумулятор еще не поступает, поэтому последний содержит пока только первое частичное произведение (рис. 7.3).

В момент появления следующего тактового импульса правый двоичный разряд множителя подключается к схеме управления сложением АЛУ. Если цифрой правого двоичного разряда множителя снова будет 1, то сдвинутое множимое прибавится к содержимому

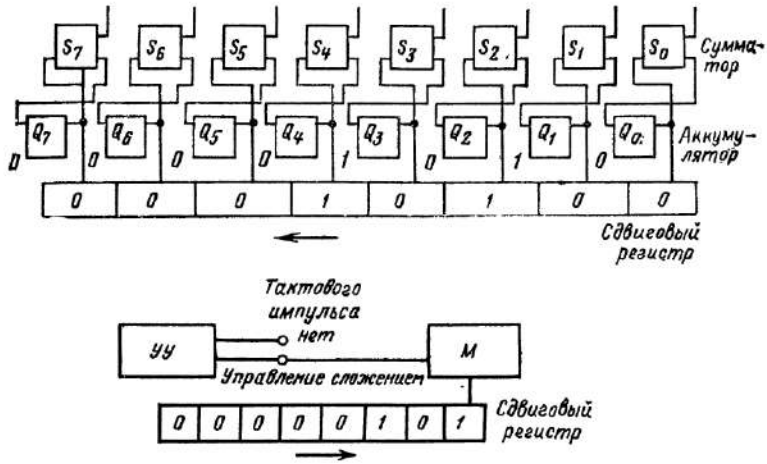


Рис. 7.3. Регистры множимого и множителя после сдвига на один разряд

аккумулятора (второе частичное произведение). Эта процедура повторяется для каждой цифры множителя, после чего в аккумуляторе появится окончательное произведение. Если для микропроцессора отдельной команды умножения не предусмотрено, описанная последовательность выполнения процедур должна быть организована с учетом имеющихся возможностей суммирования и сдвига, содержащихся в подпрограмме в виде соответствующих команд.

При умножении очень больших чисел используются аналогичные процедуры с применением арифметики с плавающей запятой.

Деление выполняется путем повторного вычитания и сдвига подобно тому, как это делается при умножении. При вычитании используются дополнительный код и процедуры, аналогичные обычным действиям карандашом на бумаге (в двоичной системе производится заем 2 вместо 10 в десятичной системе счисления). В приводимом ниже примере иллюстрируются особенности выполнения этих дейст-

вий, включая процедуру проверки отрицательности остатка, используемую при делении.

Пусть требуется разделить число 01101110 на число 1011. Представим это в такой форме: 1011 | 01101110
и выполним следующие действия:

Занимаем 2 (обычное вычитание):

```
          0
      ----------
1011 | -01101110
       -1011
      ----------
       1011
```

Находим дополнительный код делителя 1011:

```
+0100
    1
------
 0101
```

```
          0
      ----------
1011 | +01101110
       +0101
      ----------
       1011
```

Проверка отрицательности остатка (остаток отрицательный, поэтому в расчет его не принимаем, сдвигаем делитель и проверяем снова), записываем в ответ 0.

```
          00001
      ----------
1011 | -01101110
       -1011
      ----------
       0010
```

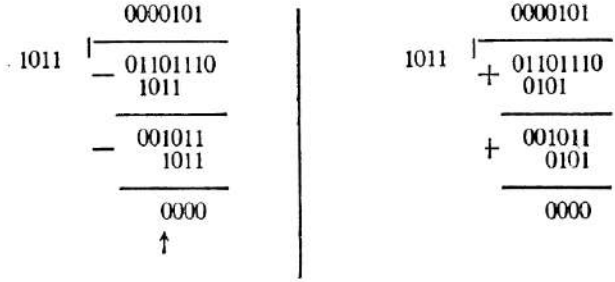
```
          00001
      ----------
1011 | +01101110
       +0101
      ----------
       0010
```

Проверка отрицательности (не отрицательный, поэтому сносим следующую цифру), записываем в ответ 1.

```
          000010
      ----------
1011 | -01101110
       -1011
      ----------
      -00101
       -1011
      ----------
       1010
```

```
          000010
      ----------
1011 | +01101110
       +0101
      ----------
      +00101
       +0101
      ----------
       1010
```

Проверка отрицательности (отрицательный, поэтому сдвигаем делитель), записываем в ответ 0.



Проверка отрицательности (не отрицательный, поэтому записываем в ответ 1).

Поскольку на этом шаге остаток нулевой, получаем ответ, в котором все последующие цифры должны быть нулями. Ответ: 00001010.

Если произвести умножение, то можно убедиться в том, что 1011×1010 действительно равно 01101110, так что получился правильный ответ.

Что такое арифметика с плавающей запятой и как она позволяет оперировать с большими и дробными числами?

В арифметике с плавающей запятой все числа масштабированы, т. е. представлены в форме $a \times 2^b$, где при умножении и делении a меньше 1_{10} и называется мантиссой представляемого числа, а b — порядком масштаба представляемого числа (всегда целое число). Поскольку $a < 1_{10}$, его можно воспринимать как дробную часть числа. Такое представление чисел в машине называется представляемым с плавающей запятой[1], так как при изменении величины порядка b десятичная запятая «плавает» (меняет позицию) в дробной части числа.

Например, двоичное число 10111111000_2 не может быть непосредственно введено в 8-разрядный регистр. Но его можно представить как $10111111 \times 2^3_{10}$, или как $1011{,}1111 \times 2^7_{10}$, или как $0{,}1011\,1111 \times 2^{11}_{10}$. (Заметим: в целях простоты масштабы чисел даны в десятичной системе счисления.) Аналогично очень маленькое двоичное число $0{,}0000\,10111111$ можно представить как $0{,}10111111 \times 2^{-4}_{10}$.

---

[1] Или, иначе, представлением в полулогарифмической форме (в разрядной сетке машины записываются мантисса и порядок числа). Порядок определяет число разрядов, отводимых в разрядной сетке для записи целой части представляемого числа. Мантиссу, удовлетворяющую условию $S^{-1} \leqslant |a| < 1$, где S — основание системы счисления, называют нормализованной. *(Прим. ред.)*

Для хранения чисел в такой форме регистры приходится подразделять на две части: одна — для хранения мантиссы, другая — для хранения порядка числа. Кроме того, необходимо выделить еще два двоичных разряда для представления знака, по одному разряду на каждый знак для мантиссы и порядка. Число двоичных разрядов, необходимых для хранения мантиссы и порядка, определяется требуемой точностью. В общем случае принято использовать более одного регистра и более одной ячейки памяти для обработки мантиссы, содержащей более восьми двоичных разрядов. При этом обработка осуществляется так же, как при рассматривавшихся ранее сложении и вычитании 16-разрядных чисел. Каждое 8-разрядное слово хранится в соседних ячейках памяти.

Умножение (деление) чисел в операциях с плавающей запятой осуществляется путем умножения (деления) их мантисс и сложения (вычитания) их порядков.

Что означает одинарная и многократно увеличенная точность выполнения арифметических операций?

Если необходима высокая точность выполнения арифметических операций, можно пользоваться несколькими регистрами записи и обработки чисел. При одинарной точности предполагается использование единственного регистра или ячейки памяти для каждого числа. Микро-ЭВМ могут быть запрограммированы таким образом, чтобы для каждого числа можно было использовать два или более регистров и ячеек памяти. Это обеспечивает повышенную точность вычислений, и такая их организация называется многократно увеличенной точностью выполнения арифметических операций (multiple precision arithmetic)[1].

При вычислениях с удвоенной точностью используются два регистра.

ГЛАВА ВОСЬМАЯ

Память

Как работает память и что подразумевается под памятью с поразрядной выборкой и памятью с пословной выборкой?

Память микро-ЭВМ состоит из элементарных полупроводниковых ячеек, организованных в некий массив. Каждая элементарная ячейка памяти способна хранить один двоичный разряд информации (0 или 1). Элементарные ячейки внутри микросхемы памяти любого типа

[1] Вопросы точности выполнения арифметических операций подробно обсуждаются в [5]. *(Прим. пер.)*

могут быть организованы одним из следующих двух способов. Первый способ состоит в том, что каждая отдельная элементарная ячейка памяти адресуется индивидуально, что означает, что в нее отдельно может быть записана и из нее может быть считана информация (организация с поразрядной выборкой информации). При другом способе для адресации выделяется вполне опре-

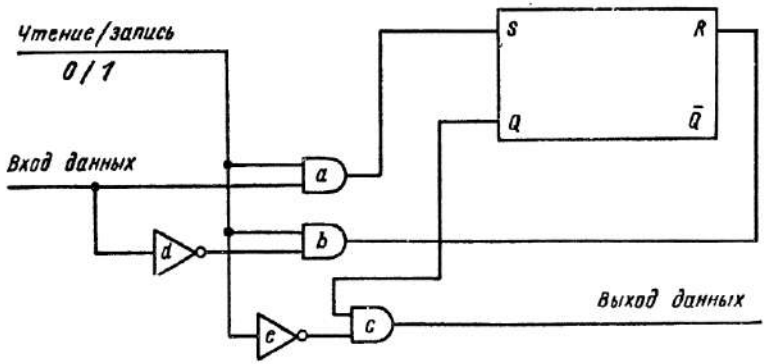


Рис. 8.1. Схема управления элементом памяти

деленное, жестко фиксированное число элементарных ячеек, в которые может одновременно записываться и из которых одновременно считываться определенная порция информации (организация с пословной выборкой информации). Типовые полупроводниковые микросхемы памяти могут содержать от 256 до 16 384 отдельных двоичных элементов для хранения информации в одном корпусе с двухрядным расположением выводов.

Как было показано ранее, триггер может использоваться в качестве временного элементарного запоминающего устройства в регистрах; посмотрим, как он может использоваться в качестве элемента памяти в ОЗУ.

Известно, что каждый элемент памяти — это обычный *RS*-триггер. На рис. 8.1 показаны логические цепи управления таким элементом, имеющиеся в любой микросхеме памяти. С их помощью путем подачи соответствующих входных сигналов можно либо посылать данные в триггер для записи, либо производить выборку информации из него.

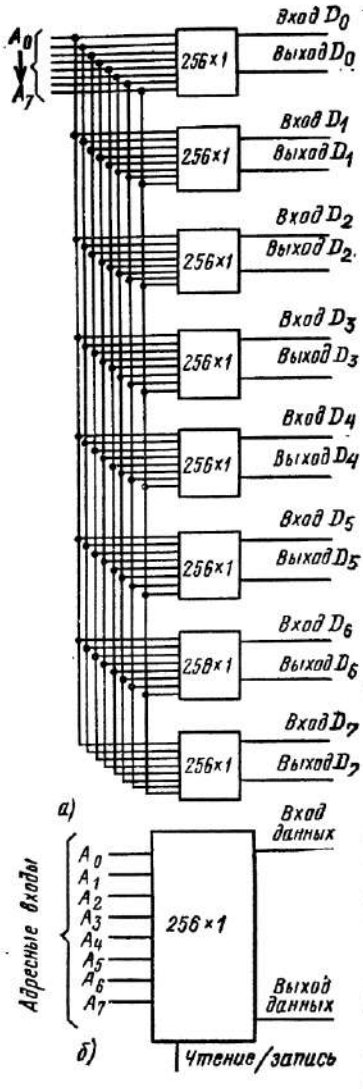


Рис. 8.2. Память объемом 256 8-разрядных слов:

а — схема соединения микросхем с поразрядной выборкой информации; *б* — схемное обозначение такой микросхемы

Рассмотрим, как работает схема управления элементом памяти, изображенная на рис. 8.1. Если в канале управления чтение/запись появился положительный потенциал U_+, то он поступит на один из входов каждого из двух элементов И (элементов *а* и *b*). Поэтому, если в это же самое время в линии передачи данных имеется сигнал U_+, то на выходе элемента *а* также появится сигнал U_+, а на выходе элемента *b* будет сформирован нулевой сигнал, так как схема НЕ (элемент *d*), реализующая операцию отрицания, изменит значение сигнала 1 на значение 0.

Следовательно, на входе, устанавливающем триггер в состояние 1 (на входе S), появится сигнал U_+, а на входе, устанавливающем триггер в состояние 0 (на входе R),— нулевой сигнал. В результате триггер окажется установленным в состояние 1 и будет хранить логическую 1. В течение всего режима записи сигнал в канале управления чтение/запись, соответствующий логической 1, инвертируется элементом НЕ (элемент *е*). Поэтому выход Q триггера оказывается заблокированным с помощью схемы И (элемент *с*), на нижнем входе которой присутствует нулевой сигнал.

Аналогично, если в линии передачи данных появится нулевой сигнал, а сигнал в канале управления чтение/запись будет тем же самым (соответствующим режиму записи), триггер установится в состояние 0 и будет его сохранять.

Чтобы можно было считать информацию из элемента памяти, в управляющем канале чтение/запись должен установиться сигнал 0. При этом заблокируются две схемы И (элементы *а* и *b*), предохранив тем самым данные от нежелательных изменений в процессе считывания, и откроется схема совпадения *с*, на нижнем входе которой появится сигнал 1 после инвертирования элементом *е* нулевого сигнала

в канале управления. На выходе элемента *с* появится сигнал, равный по значению сигналу на выходе Q триггера, т. е. сигнал, соответствующий логической 1, если триггер установлен в состояние 1, и нулевой сигнал в противном случае (когда триггер не установлен в состояние 1 и $Q=0$).

В большинстве случаев традиционные микро-ЭВМ имеют 8-разрядные шины данных. Поэтому в каждой ад-

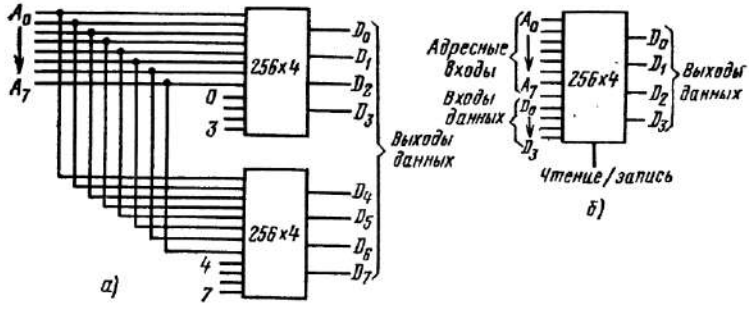


Рис. 8.3. Память объемом 256 8-разрядных слов:

а — схема соединения двух микросхем с пословной выборкой информации; *б* — схемное обозначение такой микросхемы

ресуемой (не элементарной) ячейке памяти должны храниться 8-разрядные порции информации (8-разрядные слова). Для этой цели потребуется восемь микросхем, линии передачи адреса которых будут запараллелены (как это показано на рис. 8.2, *а*), если для организации памяти микро-ЭВМ воспользоваться микросхемами с поразрядной выборкой информации. Отдельные выходы данных (8-разрядного слова) D_7, D_6, D_5, D_4, D_3, D_2, D_1, D_0 (так же, как и входы) выбираются по одному из каждой микросхемы (рис. 8.2, *б*).

Если каждая микросхема содержит 256 элементарных ячеек (элементов) памяти, получим общий объем памяти 256 слов, по восемь разрядов каждое, что потребует при адресации восемь адресных линий (от A_0 до A_7). Если бы каждая микросхема содержала 1024 элемента памяти, это дало бы 1 Кбайт памяти, т. е. 1024 8-разрядных слова, и потребовало бы десять адресных линий (от A_0 до A_9).

Вместо этого можно было бы воспользоваться микросхемами памяти с пословной выборкой информации и соединить их подобным образом. В этом случае получилась бы схема памяти (представленная на рис. 8.3, *а*) то-

го же объема, что и предыдущая, т. е. 256 8-разрядных слов (1/4 Кбайта), при использовании всего двух микросхем по 256 4-разрядных слов (схемное обозначение такой микросхемы приведено на рис. 8.3, *б*).

Память большего объема можно получить из микросхем с так называемым отпирающим входом *CE* (Chip

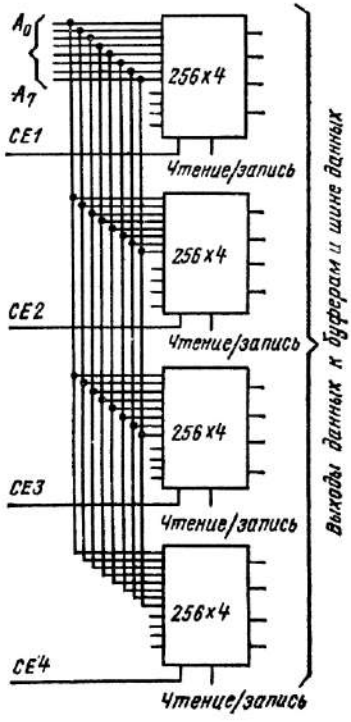


Рис. 8.4. Память объемом 1024 4-разрядных слов на четырех микросхемах с отпирающим входом

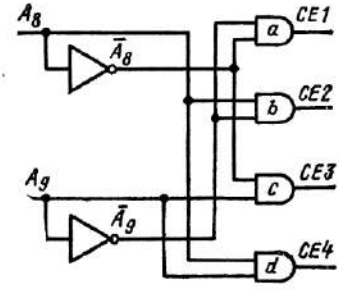


Рис. 8.5. Схема формирования сигналов для отпирающих входов микросхем

Enable). Если на этом входе появляется положительный потенциал U_+, в микросхему памяти можно записывать и считывать из нее информацию, но когда на входе *CE* присутствует нулевой потенциал, запись и считывание становятся невозможными.

На рис. 8.4 приведена схема памяти общим объемом 1024 4-разрядных слов из четырех микросхем, по 256 4-разрядных слов каждая. Схема построена благодаря использованию отпирающих входов *CE*, несмотря на то, что каждая микросхема имеет всего восемь адресных входов.

Поскольку каждая микросхема обладает памятью 256 4-разрядных слов, одноименные адресные линии четырех микросхем соединены вместе. Поэтому сигналы адреса, передаваемые по линиям A_0, A_1, A_2, A_3, A_4, A_5, A_6, A_7, адресуют одно и то же слово в каждую из четырех микросхем одновременно. Однако отпирающие входы *CE* каждой микросхемы не соединены между собой. Поэтому только одна микросхема из четырех, а именно та, у которой на отпирающем входе появится положительный потенциал U_+, действительно откроется для записи или считывания информации. Например, если необходимо передать информацию по адресу 14 (двоичное слово $0000\ 1110_2$) в микросхему с номером 3, то для этого потребуются следующие адресные сигналы:

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	*CE1*	*CE2*	*CE3*	*CE4*
0	0	0	0	1	1	1	0	0	0	1	0

Как формируются сигналы для отпирающего входа микросхемы?

Рассмотрим еще раз схему памяти объемом 1024 4-разрядных слов. Для адресации 1024 слов требуется десять разрядов (линий адреса). Из них первые восемь линий (от A_0 до A_7) соединены с микросхемами, как показано на рис. 8.4, а остальные две (A_8 и A_9) используются в схеме формирования сигналов для отпирающих входов микросхем *CE1—CE4*.

На рис. 8.5 приведена схема формирования этих четырех сигналов с использованием сигналов, поступающих от линий A_8 и A_9 адресной шины. Нетрудно заметить, что в этой схеме сигнал на выходе *CE1* получит положительный потенциал U_+ только в том случае, если этот же сигнал появится в цепях A_3 и A_9, т. е. при $A_8=A_9=0$. При этом на выходах *CE2—CE4* появится нулевой сигнал. Тогда для всех 256 номеров адреса

	A_9A_8,	$A_7A_6A_5A_4$	$A_3A_2A_1A_0$
с	0 0,	0 0 0 0	0 0 0 0
	A_9A_8,	$A_7A_6A_5A_4$	$A_3A_2A_1A_0$
по	0 0,	1 1 1 1	1 1 1 1

включительно будут выбраны только 256 слов, хранимых в первой микросхеме. Для следующих 256 номеров адреса с 01,0000 0000 по 01,1111 1111 включительно будут выбраны только 256 слов, хранимых во второй микросхеме, поскольку сигналы $A_8=1$, $A_9=0$ на входах схемы на рис. 8.5 приведут к появлению выходных сигналов $CE1=0$, $CE2=U_+$, $CE3=0$, $CE4=0$, так как в этом случае оба входа

схемы И (элемента *b*) получат положительные потенциалы U_+ ($A_8=U_+$, $\overline{A}_9=U_+$).

Аналогично для номеров адреса, начиная с 10,0000 0000 по 10,1111 1111 включительно, выбираются только 256 слов, хранимых в третьей микросхеме, так как сигналы $A_8=0$, $A_9=1$ на входах схемы на рис. 8.5 приведут к появлению выходных сигналов $CE1=0$, $CE2=0$, $CE3=U_+$, $CE4=0$. И наконец, для последней группы номеров адресов, с 11,0000 0000 по 11,1111 1111 включительно, будут выбраны только 256 слов, хранимых в четвертой микросхеме. В этом случае сигналы на входах схемы $A_8=A_9=1$ приведут к появлению выходных сигналов $CE1=0$, $CE2=0$, $CE3=0$, $CE4=U_+$.

Память объемом 1024 8-разрядных слов можно получить путем параллельного включения двух микросхем по 256×4 разрядов каждая. При этом каждая микросхема, показанная на рис. 8.4, включается так, как это показано на рис. 8.3, *а*. Системы памяти большего объема можно разрабатывать на этом же принципе, используя микросхемы с отпирающим входом.

Некоторые микросхемы памяти имеют более одного отпирающего входа, и это существенно упрощает задачу построения систем памяти с большим объемом запоминаемой информации. Например, хотя микропроцессор с 16-разрядной адресной шиной способен производить адресацию только 64 Кбайт памяти, объем памяти можно увеличить путем использования нескольких блоков по 64 Кбайт каждый, запараллеленных по адресным линиям, с обеспечением доступа к каждому из них путем включения определенной комбинации отпирающих входов микросхем по командам ввода-вывода из микропроцессора. Подобные схемы называются переключателями блоков памяти.

К числу последних разработок микросхем памяти относится изделие 5101, выполненное по КМОП-технологии[1]. Оно представляет собой статическое ОЗУ, выпускаемое в корпусе с двухрядным расположением 22 выводов, с объемом памяти 1024 двоичных разрядов, с низким уровнем потребляемой мощности и с единственным уровнем напряжения питания (5 В). Микросхема предназначена для пословной выборки 256 слов длиной по четыре двоичных разряда каждое. Элементарная ячейка памяти в микросхеме такого типа состоит из двух базовых схем инверторных переключателей, соединенных вместе по типу *RS*-триггера (см. гл. 6).

[1] Ко времени выхода оригинала книги. В настоящее время основные характеристики памяти существенно улучшены. (*Прим. ред.*)

Типовая микросхема памяти, организованная по образцу изделия 5101, содержит 1024 элементарные ячейки, размещенные в виде матрицы из 32 строк и 32 столбцов. Из восьми имеющихся адресных входов пять (A_4, A_3, A_2, A_1, A_0) могут использоваться для адресации любой из 32 строк, поскольку двоичное число 11111_2 соответствует десятичному числу 31_{10} и вместе с двоичным адресом 00000 полное число всех возможных различных адресов составит 32_{10}:

00000	00001	00010	00011	00100 . . .	11101	11110	11111
1	2	3	4	5	30	31	32

Когда сигналы кода адреса строк поступают на эти пять адресных входов ($A_4A_3A_2A_1A_0$), они проходят через буфер и поступают далее на дешифратор строк, формирующий сигнал U_+ на одной из 32 линий, относящейся к выбранной строке в соответствии с кодом адреса.

Все 32 столбца разбиты на восемь групп, по четыре столбца в каждой, и любая из этих восьми групп может адресоваться с использованием оставшихся трех адресных входов (A_7, A_6, A_5), поскольку двоичное число 111_2 соответствует десятичному числу 7_{10}, и вместе с двоичным адресом 000 полное число возможных адресов будет равно восьми.

Когда сигналы кода адреса группы столбцов поступают на три оставшихся адресных входа (A_7, A_6, A_5), они после буфера попадают на входы дешифратора групп столбцов, формирующего сигнал U_+, поступающий к ячейкам выбранной (в соответствии с входным кодом) группы столбцов из восьми возможных. Таким образом определяется единственный адрес для четырех элементарных ячеек в массиве памяти в момент появления сигналов на выходах дешифраторов строк и групп столбцов. Далее управляющий сигнал чтение/запись определяет, будет ли производиться считывание по данному адресу или запись в требуемые ячейки памяти. Таким образом, с использованием 5-разрядного адреса строки и 3-разрядного адреса столбца можно локализовать любое из 256 4-разрядных слов, размещенных в матричном массиве микросхемы памяти.

В действительности схемы выбора, а затем считывания или записи в отобранные ячейки памяти сильно различаются в разных типах микросхем.

Как соединены входы и выходы данных отдельных ячеек микросхемы?

Входы D_0 первых ячеек в каждой группе столбцов каждой строки соединены вместе и подсоединены ко входу D_0 микросхемы. Входы остальных ячеек соединены точно так же с соответствующими внешними входами микросхемы. Данные, поступающие на входы микросхемы, переписываются только в те ячейки, строки и столбцы которых одновременно получают разрешение на запись с выходов соответствующих дешифраторов под воздействием адресных сигналов.

Все выходы D_0 первых ячеек каждой группы столбцов и каждой строки объединены по схеме ИЛИ и соединены с выходом D_0 микросхемы. Другие выходы данных остальных ячеек связаны точно так же с соответствующими выходами микросхемы. Данные считываются только из тех ячеек, строки и столбцы которых одновременно получают на это разрешение.

Как микросхемы соединены с шиной данных?

Подсоединение микросхем памяти к шине данных можно осуществить с помощью многовходовых элементов ИЛИ. Однако некоторые микросхемы имеют в своем составе специальные буфера с тремя состояниями выхода — Tri-State Output (TSO) buffers (тристабильные буфера). Tri-State — это торговая марка фирмы National Semiconductors Ltd. Такие буфера освобождают от необходимости использовать элементы ИЛИ и дают возможность запараллелить все микросхемы памяти при подсоединении их к шине данных. Выход с тремя состояниями означает, что в каждый момент времени состояние выхода может характеризоваться либо появлением положительного потенциала U_+, соответствующего логическому сигналу 1, либо появлением нулевого потенциала, соответствующего логическому сигналу 0, либо высоким полным электрическим сопротивлением. В последнем случае из микросхемы считывание не производится, что равносильно отсоединению ее выхода от шины данных[1].

[1] Именно в этом смысле ранее говорилось о том, что буфер может обеспечивать электрическую изоляцию цепей. *(Прим. пер.)*

Как реализуются эти три состояния?

Они реализуются в соответствующей КМОП-структуре буфера с тремя состояниями, логика работы которого такова, что в дополняющей паре p-канального и n-канального полевых транзисторов в один и тот же момент времени либо один из них может быть включенным (этот случай уже рассматривался в гл. 1), либо оба могут быть отключены (рис. 8.6, *а*).

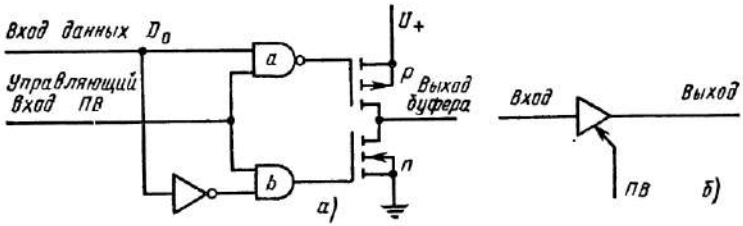


Рис. 8.6. Буфер с тремя состояниями (тристабильный буфер):
а — схема КМОП-структуры буфера; *б* — его схемное обозначение

На каждую линию выхода данных требуется один такой буфер с тремя состояниями. Если на специальном управляющем входе, обозначаемом *ПВ* (подключение выхода), появляется положительный потенциал U_+, выход буфера соединяется с линией выхода данных, являющейся его входом. Если на управляющем входе *ПВ* имеется нулевой потенциал, то между выходом буфера и его входом сопротивление становится очень высоким, что равносильно отключению выхода буфера от его входа (от линии выхода данных).

Если на линии выхода данных есть нулевой сигнал и требуется, чтобы этот сигнал появился на выходе буфера в тот момент, когда на управляющем входе *ПВ* присутствует положительный потенциал U_+, то это может произойти в том случае, если n-канальный МОП-транзистор будет открыт (т. е. на его затворе будет положительный потенциал U_+), а p-канальный МОП-транзистор будет заперт (т. е. на его затворе также будет положительный потенциал U_+).

Тогда в рассматриваемом случае на входах схемы И-НЕ (элемент *а*) будут присутствовать сигналы 0 и U_+, поэтому на выходе схемы сформируется сигнал U_+. На оба входа схемы И (элемента *b*) также поступят сигналы U_+, поэтому на выходе этой схемы появится сигнал U_+. Таким образом, p-канальный МОП-транзистор будет заперт, а n-канальный МОП-транзистор открыт, и, следовательно, на выходе буфера будет нулевой сигнал.

Если на линии выхода данных появится сигнал U_+, а сигнал на управляющем входе *ПВ* не изменится (U_+), то *p*-канальный МОП-транзистор будет открыт, а *n*-канальный МОП-транзистор заперт. Следовательно, выход буфера получит положительный потенциал U_+.

Если в какой-либо момент времени сигнал на управляющем входе *ПВ* станет нулевым, то будет совершенно безразлично, какой сигнал присутствует в этот момент на линии выхода данных. На выходе элемента *a* в этом случае будет положительный потенциал U_+, а на выходе элемента *b* нулевой. Поэтому оба МОП-транзистора будут заперты.

Сопротивление запертого полевого МОП-транзистора, как отмечалось в гл. 1, составляет около миллиона миллионов ом (10^{12}), поэтому его выход практически можно считать отсоединенным от шины данных.

Как работают адресные дешифраторы?

Применение двоичной адресации обусловлено тем, что это сокращает число необходимых внешних линий адресов. Например, 10-канальный двоичный адресный вход в микросхему памяти позволяет определить 1024 различных адреса в этой микросхеме.

Когда двоичный адрес через адресную шину попадает в микросхему памяти, его необходимо дешифрировать, чтобы обеспечить формирование правильных сигналов выбора требуемых строки и столбца в матрице памяти. Дешифрирование выполняется довольно просто с помощью схемы из логических элементов, подобной той, которая использовалась при формировании сигналов для отпирающих входов микросхем (см. рис. 8.5).

В микросхеме памяти адрес, как правило, прежде всего подается на триггерный регистр, что позволяет произвести фиксацию сигналов адреса для дальнейшего его использования (адрес может быть предоставлен дешифратору в течение более длительного промежутка времени по сравнению со временем его нахождения в адресной шине). Кроме того, на инверсных выходах $\overline{Q}_j$ регистра можно получить обратный код адреса. Микросхемы памяти, не содержащие триггерных регистров для приема адресных сигналов, должны иметь дополнительные элементы НЕ для формирования инверсных сигналов, используемых дешифратором кода адреса.

Рассмотрим в качестве примера простой случай адресации матрицы памяти размерности 4×4 (четыре строчки и четыре столбца) с 16 двоичными ячейками памяти. Для этого потребуются два адресных входа для получе-

ния четырех кодов адресов строк: 00, 01, 10, 11 и два адресных входа для получения аналогичных кодов адресов столбцов. Все это позволит произвести адресацию 16 1-разрядных ячеек памяти. Сигналы выбора строки и столбца могут быть получены дешифрированием двоичных адресных кодов схемой, представленной на рис. 8.7[1].

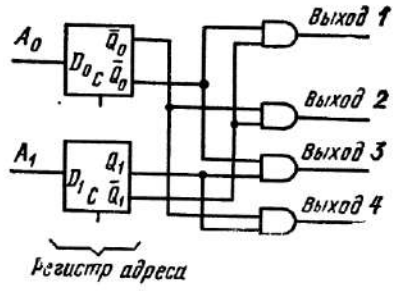


Рис. 8.7. Схема двухвходового дешифратора адреса

Связи тактового генератора с адресным регистром на схеме не показаны. Тактовые импульсы можно передавать по цепям управления таким образом, чтобы они подавались на регистр только тогда, когда требуется изменить хранящийся в нем адрес.

Схема на рис. 8.7 работает так же, как и ранее описанная схема формирования сигналов для отпирающих входов микросхем (см. рис. 8.5). Можно встретить и другие схемы дешифраторов, выполненные с применением различных логических элементов. Например, в микросхеме памяти 5101 используются элементы И для дешифрирования строки и элементы ИЛИ-НЕ для дешифрирования 3-разрядного адреса одного из восьми столбцов.

Ранее упоминалось о сигналах выбора микросхемы памяти. Они используются в цепях включения адресных дешифраторов тех микросхем, которые выбраны.

Что подразумевается под памятью статической, динамической и памятью, не сохраняющей информацию при отключении напряжения питания?

Память, о которой до сих пор шла речь, предназначена для хранения информации, не нуждающейся в непрерывном обновлении, и поэтому она называется статической. Кроме того, эта память характеризуется еще и тем,

[1] Как следует из этой схемы, дешифратор является преобразователем двоичного кода в код «один из n», под которым подразумевается n-разрядный код, каждая кодовая комбинация которого содержит одну единицу и n—1 нулей. *(Прим. пер.)*

что хранимая в ней информация теряется при выключении напряжения питания. Поэтому эта память относится к памяти, не сохраняющей информацию при отключении напряжения питания, или непостоянной памяти (volatile memory).

Наиболее распространенным элементом ОЗУ является микросхема 2102 с 16 выводами, в корпусе с двухряд-

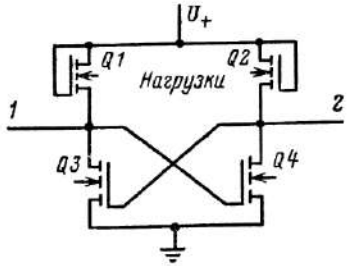


Рис. 8.8. Базовая ячейка микросхемы памяти 2102, представляющая собой триггер из двух инвертирующих n-МОП-ключей

ным расположением выводов, с объемом памяти 1024×1 двоичных разрядов, изготовленная по n-МОП-технологии. В ней для хранения информации используются схемы из n-МОП-транзисторов с перекрестным соединением, причем каждая элементарная ячейка состоит из четырех таких транзисторов (рис. 8.8).

Данные хранятся в виде положительных зарядов на затворах транзисторов либо Q_3, либо Q_4, что и приводит к отпиранию соответствующего МОП-транзистора. Транзисторы Q_1 и Q_2 имеют повышенное сопротивление по сравнению с обычным МОП-транзистором и используются в схеме как нагрузки.

Предположим, что транзистор Q_3 открыт, т. е. на его затворе сохраняется положительный заряд. Через поддерживаемый в открытом состоянии транзистор Q_1 ток может протекать к земле. В связи с этим потенциал на входе *1* близок к нулю. (Действительное его значение зависит от состояния сопротивлений проводящих транзисторов Q_1 и Q_2). Следовательно, потенциал затвора транзистора Q_4 нулевой или близок к нулю, и, таким образом, транзистор Q_4 заперт. Заряд на затворе транзистора Q_3 поддерживается транзистором Q_2 путем возмещения той части заряда, которая могла бы стечь. Поэтому ячейка памяти остается в том же логическом состоянии до момента, пока оно не сменится в результате новой записи. Все это остается в силе и в случае, когда транзистор Q_4 открыт, а Q_3 заперт.

Другой распространенный тип ячейки ОЗУ — это ячейка динамического ОЗУ, в которой данные хранятся в виде зарядов конденсатора. На рис. 8.9 приведена схема ячейки динамической памяти на трех n-МОП-транзисторах.

Полевой транзистор Q_4 является общим для всех ячеек памяти столбца матрицы и служит для предварительной зарядки конденсатора C_D. Чтобы иметь возможность считать информацию из ячейки, конденсатор C_D сначала предварительно заряжается до напряжения, очень близкого к U_+, через транзистор Q_4, затвор кото-

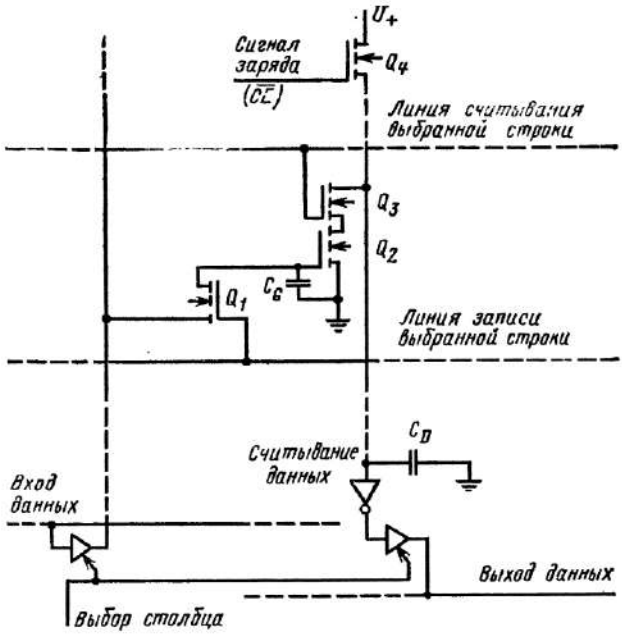


Рис. 8.9. Ячейка динамической памяти на *n*-МОП-транзисторах

рого подсоединен к линии передачи сигнала $\overline{CE}$ (Not Chip Enable). На линии считывания, общей для ячеек строки матрицы, должен присутствовать положительный потенциал U_+, открывающий транзистор Q_3. Если потенциал заряда, хранимого на конденсаторе C_G, соответствует логической 1 (т. е. очень близок к U_+), транзистор Q_2 открывается, и конденсатор C_D разряжается через транзисторы Q_3 и Q_2 на землю. Однако, если напряжение на конденсаторе C_G соответствует логическому 0 (нулевое), транзистор Q_2 заперт, и поэтому конденсатор C_D остается заряженным до напряжения U_+. Следователь-

но, на линии считывания данных появляется обратный код хранимых данных.

Заметим, что состояние конденсатора C_G остается неизменным в течение времени считывания. Данные могут считываться из C_D в обратном коде в выходной регистр или пересылаться в шину данных через соответствующий буфер. Чтобы записать информацию в ячейку памяти, необходимо подать положительный потенциал на линию выбора строки записи вместо линии выбора строки считывания (часто организуется единственный, объединенный вход чтение/запись).

Положительный потенциал, появившийся в линии выбора строки записи (являющейся общей для ячеек строки матрицы), открывает транзистор Q_1 в каждой ячейке строки, который передает напряжение, присутствующее в линии записи данных выбранного столбца, одному из конденсаторов C_G выбранной ячейки памяти.

Сигналы выбора строки считывания и выбора строки записи для каждой строки матрицы могут быть получены путем объединения сигналов считывания и записи по схеме И с соответствующими сигналами выбора строки, формируемыми дешифратором строки.

Хотя считывание само по себе не разрушает записанную информацию, заряды на конденсаторах C_G уменьшаются со временем из-за утечки. Поэтому величину зарядов приходится поддерживать с помощью специальной восстанавливающей схемы, которая время от времени считывает содержимое ячеек и производит повторную запись в них. Такая схема включена, например, в состав микросхемы микропроцессора Z80, но чаще поставляется отдельно.

Как устроена постоянная память?

Постоянное запоминающее устройство (ПЗУ) (ROM — Read Only Memory) можно представить себе в виде матрицы постоянно разомкнутых или замкнутых контактов, пропускающих ток только в одном направлении (рис. 8.10).

Следует обратить внимание на то, что это устройство довольно похоже на ОЗУ, однако в нем отсутствуют цепи записи данных. Линии выбора строки и столбца получают сигналы от адресных линий так же, как и в ОЗУ,

а адресация к отдельным ячейкам и считывание из них во многом схожи с аналогичными процессами в ОЗУ.

Существуют различные типы ПЗУ, и их главное различие состоит в том, каким путем получены замкнутые и разомкнутые контактные соединения. В запрограммированных ПЗУ с применением специальных масок (ПЗУ с

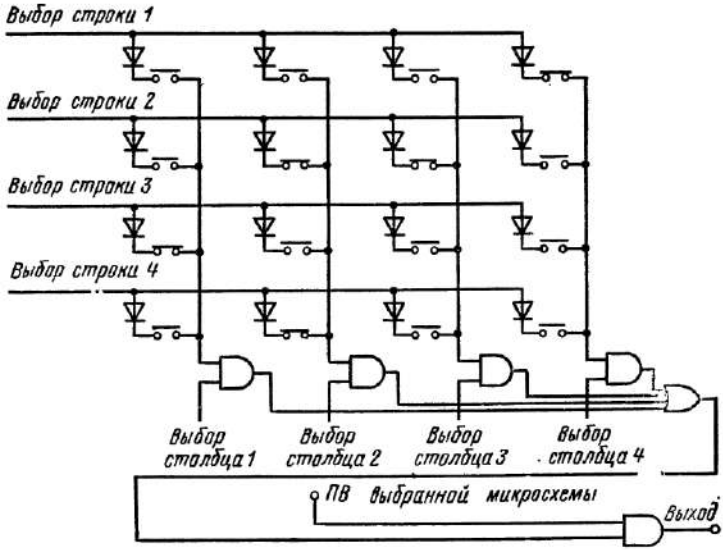


Рис. 8.10. Упрощенная схема матричного ПЗУ размерности 4×4

масочным программированием) на заключительной стадии производства оставляются только необходимые соединения, а все ненужные исключаются.

В программируемых постоянных запоминающих устройствах (ППЗУ) (PROM — Programmable Read Only Memory) контактные соединения выполнены из плавкого материала. В дальнейшем в ячейках с требуемым адресом они могут быть расплавлены (электрически программируемые ПЗУ). Это дает возможность пользователю запрограммировать устройство памяти после его изготовления в соответствии с предназначенной для хранения информацией.

Существуют ППЗУ с возможностями стирания хранимой информации и повторного программирования. В рас-

сматриваемой упрощенной модели это соответствовало бы восстановлению всех контактных соединений и затем повторному разрушению необходимых соединений, как того требовала бы новая информация, предназначенная для хранения. Такие ПЗУ называются перепрограммируемыми ПЗУ, или программируемыми ПЗУ со стиранием информации[1] (EPROM — Erasable Programmable Read Only Memory).

В настоящее время существуют два основных вида технологии изготовления микросхем памяти: биполярная технология и МОП-технология. Их основное различие — это цикл обращения к памяти (время, требующееся для того, чтобы данные по определенному адресу оказались доступными для их последующего использования). Цикл обращения к памяти, построенной по биполярной технологии, почти в 10 раз меньше времени доступа к памяти, построенной по МОП-технологии. Однако схемы из биполярных транзисторов занимают бóльшую площадь на кристалле, чем эквивалентные МОП-схемы, и выпускаются объемами памяти в 1, 2 и 4 Кбит. Микросхемы постоянной памяти на основе МОП-технологии выпускаются объемом до 16 Кбит. Перепрограммируемые ПЗУ изготавливаются только с применением МОП-технологии.

Существующие типы ПЗУ можно классифицировать следующим образом:

[1] Для стирания данных в перепрограммируемых ПЗУ используется ультрафиолетовое или рентгеновское облучение. Для записи данных — постоянное напряжение. *(Прим. пер.)*

По способу программирования различают следующие виды ПЗУ:

В первых выпусках электрически программируемых ПЗУ (ЭП ПЗУ) использовались нихромовые плавкие пе-

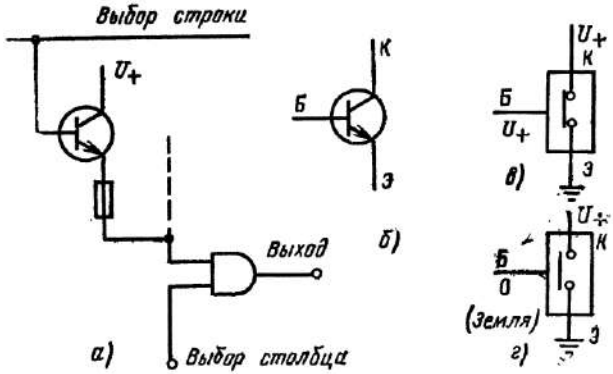


Рис. 8.11. Ячейка биполярного программируемого ПЗУ:

а — схема ячейки с биполярным транзистором и плавкой перемычкой; *б* — биполярный транзистор (*Б* — база, *Э* — эмиттер, *К* — коллектор); *в*, *г* — модели транзисторного ключа (замкнутого и разомкнутого)

ремычки[2]; поэтому для разрушения соединений в процессе программирования приходилось прибегать к сильным токам. В схему каждой элементарной ячейки памяти входили биполярный транзисторный ключ и плавкая перемычка (рис. 8.11, *а*).

Если строка матрицы памяти выбрана в процессе считывания, то на линии выбора строки появляется положительный потенциал U_+, открывающий транзисторный ключ, и, если плавкая перемычка не разрушена, по-

[1] Очевидно, автор здесь имеет в виду случай, когда вместо плавкой перемычки в схеме используются встречно включенные диоды. Программирование осуществляется путем подачи напряжения, при котором в одном из диодов происходит пробой (короткое замыкание). (*Прим. ред.*)

[2] Нихромовые плавкие перемычки представляют собой нихромовые ленточки шириной $2 \cdot 10^{-8}$ М. (*Прим. пер.*)

ложительный сигнал проходит на линию столбца, соединенную по схеме И со входом выбора столбца, в результате чего на выходе появляется сигнал U_+. Если плавкая перемычка разрушена, на выходе выбранного столбца будет нулевой сигнал.

Программируемые ПЗУ с нихромовыми плавкими соединениями обладали недостатком — самовосстановлением, т. е. по прошествии какого-то времени некоторые из разрушенных соединений вновь становились проводящими. Этого недостатка удалось избежать благодаря применению вместо нихрома поликристаллического кремния в качестве материала плавкой перемычки.

Постоянные запоминающие устройства, включая программируемые пользователем, выпускаются также с применением полевых транзисторов вместо биполярных. Однако в перепрограммируемых ПЗУ используются только МОП-транзисторы особого вида, называемые МОП-транзисторами с плавающим затвором и лавинной инжекцией (FAMOS — Floating Gate Avalance-injection Metal Oxide Semiconductor). Их можно рассматривать как p-канальные МОП-транзисторы без внешних связей с затворами.

Подача повышенного напряжения на исток или сток (—30 В) приводит к инжекции электронов с высоким уровнем энергии в результате процесса, называемого лавинной инжекцией, в затвор из истока или из стока. Появившись на затворе, отрицательный заряд электронов создает электрическое поле в канале, стремящееся открыть p-канальный МОП-транзистор (т. е. увеличить его проводимость между истоком и стоком).

Когда программирующее напряжение снимается, у электронов затвора нет пути для его разрядки, поскольку затвор окружен изолирующим слоем из двуокиси кремния (рис. 8.12, *а*). Поэтому заряд устойчиво сохраняется на затворе. Как показали эксперименты, двуокись кремния — настолько хороший изолятор, что даже по истечении 10 лет на затворе должно сохраниться более 70 % заряда. Поскольку с затвором нет соединений, электроны не могут быть удалены с него электрическими средствами.

Однако, если МОП-транзистор с плавающим затвором и лавинной инжекцией подвергнуть воздействию ультрафиолетовых лучей, электроны получат достаточную энергию, чтобы перескочить обратно на кремниевую

подложку. При этом затвор разряжается, и ячейка возвращается в свое первоначальное незапрограммированное состояние. Таким образом, перепрограммируемые ПЗУ могут программироваться и использоваться как обычные ПЗУ с масочным программированием, но позже, если потребуется, их можно перепрограммировать после воздействия ультрафиолетовыми лучами. Перепро-

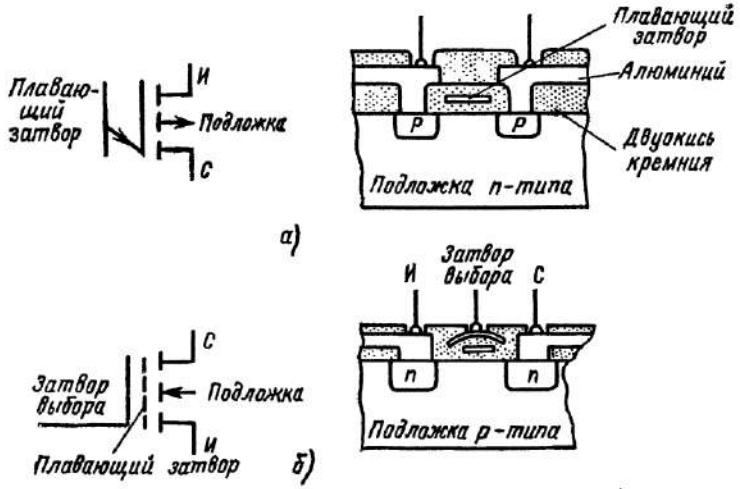


Рис. 8.12. МОП-транзисторы ячейки перепрограммируемой памяти: *а* — транзистор с плавающим затвором; *б* — транзистор с составным затвором

граммируемые ПЗУ часто бывают очень полезны при производстве какого-либо специального прототипа ПЗУ с масочным программированием[1].

Другим типом ячейки перепрограммируемого ПЗУ является однотранзисторная ячейка с так называемым составным затвором (stacked-gate), которая используется в микросхеме 2708. Составной затвор включает в себя нижний плавающий затвор и верхний затвор выбора,

[1] Изготовление маски, необходимой для построения специализированного ПЗУ, — дорогостоящая операция. Поэтому экономически оправдано сначала отладить работу системы с перепрограммируемым аналогом требуемого ПЗУ и лишь затем приступить к изготовлению более дешевого ПЗУ с масочным программированием. *(Прим. пер.)*

соединенный с линией выбора строки матрицы памяти (рис. 8.12, *б*).

Ячейка программируется лавинной инжекцией электронов в плавающий затвор. Электрическое поле, создаваемое отрицательным зарядом плавающего затвора, стремится уменьшить проводимость *n*-канального МОП-транзистора, если выбран он. Поэтому для его включения к затвору выбора требуется приложить большее положительное напряжение, чем в случае с незапрограммированной ячейкой. Это и используется для хранения данных. Чтобы под действием сигнала выбора открывались только незапрограммированные ячейки, в плавающий затвор инжектируется заряд подходящего уровня. Информацию из ячеек можно удалить при помощи ультрафиолетового облучения соответствующей частоты.

Все рассмотренные типы устройств постоянной памяти: обычные ПЗУ (с масочным программированием), программируемые (пользователем) и перепрограммируемые ПЗУ (со стиранием информации) относятся к виду памяти, сохраняющей информацию при выключении напряжения питания (non-volatile memory).

Для чего используются вспомогательные запоминающие устройства, такие как кассеты с магнитной лентой и флоппи-диски?

Внешние запоминающие устройства удобны тем, что, во-первых, они также принадлежат к типу устройств памяти, сохраняющих информацию при отключении напряжения питания, и, во-вторых, способны постоянно хранить большие объемы информации (данные и программы), готовой для последующего использования. Если решено хранить данные на кассете с магнитной лентой (т. е. организовать файл), то время доступа к этим данным будет довольно велико, особенно если требуемые данные находятся где-то в конце ленты, в таком случае придется просмотреть ее почти всю, прежде чем требуемая информация будет найдена.

От этого недостатка свободен флоппи-диск, предназначенный для записи и считывания информации в магнитной форме на диске, хранимом и используемом в специальном тонком бумажном конверте (рис. 8.13, *а*). Данные на диске располагаются последовательно на концентрических дорожках (не в виде спирали, как в механиче-

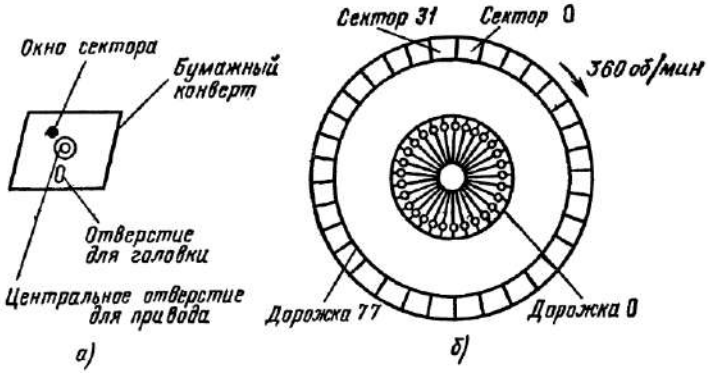


Рис. 8.13. Дисковая система IBM 3740 с объемом запоминаемой информации до 243 000 байт:

а — бумажный конверт; *б* — диск

ской звукозаписи). Каждая дорожка разделена на сегменты. Универсальная (для считывания и записи) магнитная головка, управляемая командами программного обеспечения, может быть подведена непосредственно к любому сектору каждой дорожки. Это существенно сокращает время доступа к данным по сравнению с кассетой с магнитной лентой. Однако память такого рода все-таки менее быстродействующая, чем полупроводниковая память ОЗУ и ПЗУ[1].

В дисковой системе IBM 3740 (рис. 8.13) используется по одному секторному окошку для каждого сектора (жестко размеченный диск — hard sectored disc), и локализация необходимого сектора осуществляется аппаратными средствами. Некоторые дисковые системы располагают только одним секторным окошком, и в таком случае локализация других секторов осуществляется средствами программного обеспечения (программно-размеченный диск — soft sectored disc).

Что такое память на цилиндрических магнитных доменах?

Промежуточное положение между ОЗУ и памятью на флоппи-дисках занимает другой вид магнитной памяти — так называемая память на цилиндрических магнитных

[1] Среднее время выборки информации из памяти на флоппи-диске составляет примерно 500 мс (см. [3]). *(Прим. пер.)*

доменах (ЦМД). Небольшие домены, или области намагничивания, формируются на пленке из магнитного материала. Закодированная в них информация может считываться в последовательной форме. Память на ЦМД не утрачивает информацию при выключении напряжения питания, и ее преимущество состоит в том, что относительно дешевым способом можно изготовить память для хранения больших объемов информации (более 1 млн. бит на площади всего в 1 см^2). Устройства памяти на ЦМД дают возможность пользоваться огромными объемами хранимых данных ценой невысоких скоростей записи и считывания.

Что означает страничная адресация?

Для адресации всей памяти микро-ЭВМ используются сигналы, передаваемые по адресной шине. Если используется 16-разрядная адресная шина, каждый адрес может быть представлен шестнадцатью двоичными цифрами (разрядами) или четырьмя шестнадцатеричными цифрами (см. гл. 3). Например:

1101 0010 1011 1110

D 2 B E

Первая половина адреса в таком представлении считается номером страницы, а вторая — номером ячейки в странице. Следовательно, первые восемь двоичных разрядов представляют номера $2^8=256$ страниц, каждая из которых содержит по 256 ячеек. Номера этих ячеек представлены восемью самыми младшими двоичными разрядами адреса.

Таким образом, нулевая страница содержит адреса с первыми двумя нулевыми шестнадцатеричными цифрами (рис. 8.14), т. е. адреса с 0000_{16} по $00FF_{16}$ (с 0_{10} по 255_{10}). Первая страница содержит адреса с 0100_{16} по $01FF_{16}$ (с 256_{10} по 511_{10}), вторая страница соответственно с 0200_{16} по $02FF_{16}$ (с 512_{10} по 767_{10}) и т. д. до 256-й страницы, которая содержит адреса с $FF00_{16}$ по $FFFF_{16}$ (с 65281_{10} по 65535_{10}).

Что такое карта памяти?

Это диаграмма, показывающая, каким именно из 65 536 возможных адресов (адресного пространства) соответствуют те или иные отдельные области памяти микро-ЭВМ (рис. 8.15). Карта памяти часто содержит и

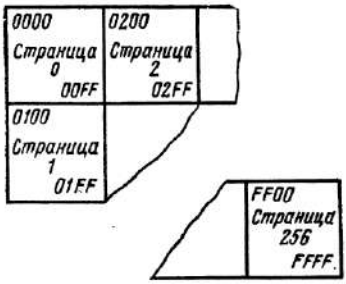


Рис. 8.14. Адресация страниц

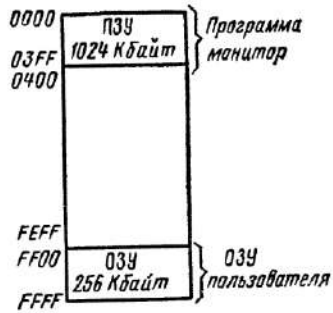


Рис. 8.15. Карта памяти

другую информацию, например, относящуюся к особенностям использования специальных адресов памяти. Важно при этом учитывать информацию о нахождении в адресном пространстве области ПЗУ, где располагается специальная программа, к выполнению которой переходит микропроцессор после сигнала «сброс» для того, чтобы обеспечить микро-ЭВМ правильный начальный запуск.

Что означает адресация портов ввода и вывода как ячеек памяти?

Это метод организации в микро-ЭВМ ввода-вывода путем адресации портов ввода-вывода так, как если бы они были ячейками памяти. Таким образом, адресация внешних устройств при этом ничем не отличается от адресации при считывании и записи в память. В некоторых системах вместо адресации портов ввода-вывода как ячеек памяти используются специальные команды ввода-вывода.

Что подразумевается под способом адресации?

Память используется для хранения кодов операций, адресов, данных, предназначенных для обработки, и ее результатов. Чтобы микропроцессор оказался в состоянии выполнить команду, ему необходимо сообщить: 1) код операции; 2) место расположения в памяти первого операнда (или код первого операнда); 3) место

расположения в памяти второго операнда (или код второго операнда); 4) место в памяти, куда нужно поместить результат.

Перечисленная в п. 1—4 информация могла бы указываться в каждой команде, но обычно это не делается из-за ограничений на длину команды. Часто место помещения данных (п. 4) подсказывается самим кодом операции, как, например, в команде LDA (ЗАГРУЗКА), в которой подразумевается, что данные нужно загрузить в аккумулятор (см. гл. 3).

Иногда пп. 2 и 3 также подразумеваются или не требуются вовсе, как, например, в рассматриваемой в гл. 6 команде INC (УВЕЛИЧИТЬ НА 1), для которой подразумевается: «число, записанное в аккумуляторе, увеличить на 1 и поместить результат обратно в аккумулятор».

Различные способы определения места расположения операнда называются способами адресации. При прямой адресации в команде указывается абсолютный (физический) адрес операнда, а при косвенной адресации абсолютный адрес находится с помощью некоторых косвенных средств. В обычном микропроцессоре используется 8-разрядный код операции и 16-разрядный код адреса, а команды занимают от 1 байта (восемь двоичных разрядов) до 3 байт. Различные микропроцессоры допускают различные способы адресации, и, к сожалению, изготовители дают иногда разные названия одним и тем же способам. Как правило, чем меньше набор команд, тем большим числом способов адресации можно пользоваться. Именно комбинация различных способов адресации с базовым набором команд определяет вычислительные возможности каждого конкретного микропроцессора.

Каковы наиболее распространенные способы адресации?

Прямая адресация. При прямой адресации адрес операнда содержится в самой команде, после кода операции. Иногда требуется использовать только 1 байт адреса, и операнд, следовательно, должен быть на нулевой странице. Команда в этом случае будет занимать 2 байта, причем первый байт отводится для кода операции, а второй — для восьми самых младших разрядов адреса.

При 16-разрядном адресе адресация иногда называет-

ся расширенной. Приведем пример команды, занимающей 3 байта:

Содержание команды	Форма команды
	Код операции — адрес
LOAD (ЗАГРУЗИТЬ) аккумулятор содержимым ячейки памяти ** **	AD ** **

Адресация аккумулятора[1]. Эта команда занимает 1 байт и всегда относится к аккумулятору. Следовательно, никаких дальнейших адресов не нужно, только код операции. Например:

Содержание команды	Форма команды
	Код операции
COMPLEMENT (ДОПОЛНЕНИЕ) Получить обратный код данных, содержащихся в аккумуляторе	17

Непосредственная адресация. В этом случае сам операнд занимает второй байт в команде. Этот вид адресации удобен, когда необходимо выполнить арифметическую или логическую операцию над константами. Например:

Содержание команды	Форма команды
	Код операции — данные
ADD (СЛОЖИТЬ) *пп* с содержимым аккумулятора (где *пп* — 8-разрядное двоичное слово данных)	6E *пп*

Косвенная адресация. В этом случае команда содержит либо адрес памяти, либо адрес регистра, содержащего адрес операнда (каждая ячейка памяти может содержать только 8-разрядное слово, поэтому, если не используется косвенная адресация через регистр, операнд может быть на нулевой странице). Например:

Содержание команды	Форма команды
	Код операции — адрес
LOAD (ЗАГРУЗИТЬ) аккумулятор данными ячейки памяти, указанной содержимым адреса ** **	3A ** **

Иногда полный 16-разрядный адрес содержится в двух соседних ячейках памяти, и в таком случае команда могла бы быть следующей: LOAD (ЗАГРУЗИТЬ) акку-

[1] Частный случай неявной адресации. *(Прим. ред.)*

мулятор данными, содержащимися в ячейках памяти, указанных знаками ** ** и ** **+1.

Индексная адресация. В этом случае адрес, занимающий второй байт команды, комбинируется (суммируется) с содержимым регистра, называемого индексным регистром, для получения адреса операнда. Этот вид адресации удобен тогда, когда одна и та же последовательность вычислений должна быть выполнена для разных наборов данных, хранящихся в последовательных ячейках памяти. Например:

Содержание команды	Форма команды
	Код операции
LOAD (ЗАГРУЗИТЬ) слово в ячейку памяти, определенную содержимым индексного регистра +8-разрядное двоичное слово ** аккумулятора	A E **

Неявная адресация. Это случай, когда команда занимает 1 байт, причем код операции в ней фиксирован, а адрес подразумевается самой командой. Команда всегда выполняет одну и ту же операцию. Например:

Содержание команды	Форма команды
	Код операции
LOAD (ЗАГРУЗИТЬ) указатель стека (регистр в микропроцессоре) содержимым регистра *X*	AF

Относительная адресация. В этом случае адрес, содержащийся во втором байте команды, суммируется с содержимым счетчика команд для получения адреса операнда. Этот способ адресации может использоваться для выполнения команд условного перехода. Иногда второй байт (смещение) представлен в дополнительном коде, и это позволяет в программе осуществить переход вперед или назад (т. е. к содержимому счетчика команд прибавить или отнять смещение). Сформированный таким образом новый адрес может находиться в диапазоне от «содержимое счетчика команд+127_{10}» до «содержимое счетчика команд —128_{10}». Например:

Содержание команды	Форма команды
	Код операции — смещение
Выбрать следующую команду из ячейки памяти по адресу, который даст содержимое регистра счетчика команд и 8-разрядное слово *dd*	10 *dd*

ГЛАВА ДЕВЯТАЯ

Управление

Что такое управляющие сигналы?

Арифметическо-логическое устройство (АЛУ) выполняет под воздействием различных управляющих сигналов те или иные операции из некоторого фиксированного множества: ADD (СЛОЖИТЬ), CLEAR (ОЧИСТИТЬ), INC (УВЕЛИЧИТЬ НА ЕДИНИЦУ) и т. п. Сигналы управления формируются управляющим устройством внутри микропроцессора. Управляющее устройство воспринимает команды в виде двоичных кодов из программы и затем формирует соответствующие развернутые во времени последовательности сигналов управления. Управляющее устройство регулирует поступление данных через устройство ввода-вывода к различным другим частям микро-ЭВМ, а также управляет процедурами записи и считывания информации из памяти. В его состав входит центральный тактовый генератор, синхронизирующий работу всех устройств микро-ЭВМ.

Линии передачи синхронизирующих (тактовых) импульсов, адресная шина данных, линии передачи сигналов для отпирающих входов микросхем, цепи питания, считывания и записи и другие управляющие цепи образуют структуру шины. Конечно, нет такой особой необходимости, чтобы абсолютно все линии шин требовались каждому устройству микро-ЭВМ, однако из чисто практических соображений удобнее направить все линии шины ко всем устройствам микро-ЭВМ и обеспечить тем самым универсальную компоновку всех связей, обеспечивающую легкий доступ устройств ко всем линиям шин. В структуре шины типовой микро-ЭВМ может содержаться от 50 до 100 отдельных линий, и они часто организуются как совокупность параллельных проводящих полосок печатной платы, называемой объединительной. Другие платы, содержащие микропроцессор, память и т. п., могут вставляться в объединительную плату с помощью торцевых разъемов (печатный разъем с контактными ламелями по краю).

Какие устройства микропроцессора участвуют в управлении и пересылке данных?

В составе устройств типового микропроцессора имеется регистр, выполняющий функции регистра адреса памяти или указателя адреса памяти. Он используется во время выполнения команды для хранения адреса ячейки памяти, сформированного соответствующим сигналом, передаваемым по адресной шине. Поскольку адрес зани-

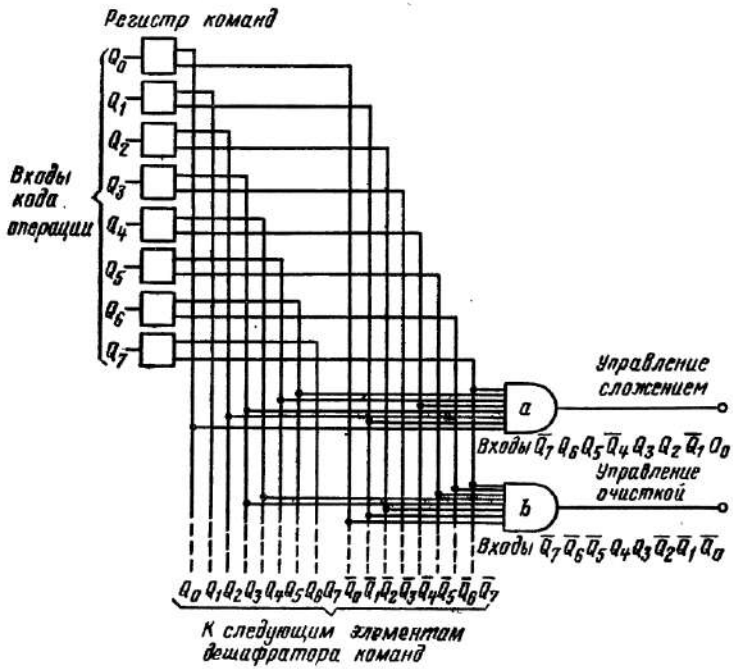


Рис. 9.1. Фрагмент схемы дешифратора команд для формирования управляющих сигналов ADD и CLEAR

мает 16 двоичных разрядов, то этот регистр 16-разрядный, либо используются два 8-разрядных регистра. Регистр — указатель адреса памяти — может использоваться также для изменения или формирования другого адреса во время выполнения команды.

Другой регистр, часто называемый буферным, используется для хранения слова, которое должно быть пере-

писано в память или которое только что считано из ячейки памяти.

Когда считывается очередная команда программы, код операции запоминается в регистре микропроцессора, называемом регистром команд. Код операции, хранящийся в регистре команд, дает распоряжение микропроцессору выполнить требуемую операцию. Дешифратор команды преобразует код операции, записанный в регистре команд, в соответствующие управляющие сигналы, и устройство управления подает их в нужной последовательности к необходимым устройствам микро-ЭВМ, чтобы поступившая команда была выполнена.

Например, если код операции для команды ADD (СЛОЖИТЬ) есть 6 D (т. е. 01101101_2), а для команды CLEAR (ОЧИСТИТЬ) — 18 (т. е. 00011000_2), то с помощью логической схемы на рис. 9.1 сформируются управляющие сигналы ADD и CLEAR тогда и только тогда, когда в регистр команд поступят соответствующие коды этих операций $Q_7Q_6Q_5Q_4Q_3Q_2Q_1Q_0$.

При коде операции $0110\ 1101_2$ на каждый вход схемы И (элемента *a*) поступит положительный потенциал U_+, и, следовательно, точно такой же сигнал появится на выходе элемента *a*. При любом другом коде операции по крайней мере на одном из входов элемента *a* появится нулевой сигнал, поэтому управляющий сигнал ADD на выходе элемента *a* сформирован не будет (на выходе элемента *a* будет нулевой сигнал). Аналогичные рассуждения справедливы и для элемента *b*, на выходе которого появится сигнал U_+ только при поступлении кода операции $0001\ 1000_2$.

Любая пересылка данных, необходимых для выполнения какой-либо команды, также невозможна без специальных управляющих сигналов, посылаемых в требуемой последовательности устройством управления. Например, в некоторых случаях для выполнения команды ADD необходимо, чтобы предназначенные для сложения числа были извлечены из памяти и переданы в арифметическо-логическое устройство. Каким образом управляющее устройство формирует специальные последовательности управляющих сигналов, требующихся для выполнения различных команд, будет объяснено несколько позже.

В ходе выполнения программы двоичное слово, хранящееся в каждой ячейке памяти, может представлять собой либо необходимые для обработки данные, либо команду, либо адрес следующей ячейки памяти. Порядок, в котором двоичные числа записываются и считыва-

ются, указывает на то, что именно они представляют собой в данный момент, и микропроцессор должен следить за этим порядком.

Например, слово 6DCB00 могло бы представлять выражение «СЛОЖИТЬ содержимое ячейки 00CB с содержимым аккумулятора». Значения первых двух шестнадцатеричных разрядов (6D) являются операционным кодом команды, а следующие четыре (CB00) — адресом в памяти. Когда этот адрес считывается, его содержимое представляет собой данные, подлежащие обработке.

Микропроцессор обычно содержит еще 1-разрядный регистр, называемый внутренним флагом. Его содержимое используется для указания, должен ли микропроцессор произвести выборку части команды из памяти или просто выполнить команду. Этот флаг иногда называется регистром выборки/исполнения команды. Если флаг выставлен (в регистре записана 1), то микропроцессор воспринимает считываемое из памяти слово как часть команды. Если флаг не выставлен (в регистре хранится 0), то слово воспринимается им как данные, подлежащие обработке. Этот флаг пользователю не доступен, поэтому рассматривать его не будем.

Существует, кроме того, еще один регистр, называемый программным счетчиком или счетчиком команд (PC — Program Counter), который содержит адрес очередной команды, подлежащей выборке. Адреса представляют собой 16-разрядные слова. Поэтому в качестве счетчика команд в микропроцессоре часто используются два 8-разрядных регистра. Один из них, называемый НИЖНИМ (LOW), предназначен для хранения восьми самых младших разрядов адреса. Он обозначается как PCL (Program Counter Low). Другой — ВЕРХНИМ (HIGH). Он предназначен для хранения восьми самых старших разрядов адреса и обозначается PCH (Program Counter High). Когда команда выполняется, программа продвигается на серию шагов во время обработки данных. Каждый такой шаг увеличивает число, хранящееся в счетчике команд, до тех пор, пока оно не укажет ячейку памяти, в которой хранится очередная команда, подлежащая выборке.

При выполнении любой программы последовательность событий примерно следующая. Сначала в счетчик команд записывается адрес пуска первой команды. Затем осуществляется выборка по этому адресу из памя-

ти содержимого ячейки, воспринимаемого как операционный код команды. Коды операции помещаются в регистр команд, соединенный со схемой дешифратора команд, на выходе которого будет сформирован соответствующий управляющий сигнал. Далее произойдет приращение содержимого счетчика команд, и если код операции указывает на прямой способ адресации, то осуществляется выборка содержимого из этой ячейки памяти, и оно помещается в младшие восемь разрядов регистра указателя адреса памяти. Снова происходит приращение содержимого счетчика команд, и, если код операции указывает на трехбайтную команду, содержимое ячейки по адресу, данному теперь счетчиком команд, извлекается из памяти и помещается в восемь старших разрядов регистра указателя адреса памяти. Выборка (вызов из памяти) команды завершена.

Содержимое ячейки по адресу, записанному в регистре указателя адреса памяти, считывается в буферный регистр, после чего выполняется операция, соответствующая коду операции, который все еще хранится в регистре команд. Содержимое счетчика команд получает приращение, и производится выборка слова, находящегося в ячейке памяти по адресу, указанному теперь в счетчике команд. Это слово воспринимается как операционный код очередной команды. Описанный процесс повторяется до тех пор, пока не встретится команда останова программы[1].

Точная последовательность событий при выполнении программы, естественно, варьируется в зависимости от используемых команд и разных способов адресации. Содержимое счетчика команд может увеличиваться или уменьшаться более чем на один адрес команды, и программе в этом случае придется осуществлять переходы к другим операциям заданной последовательности. Таким образом, последовательность выполнения команд программы можно менять в зависимости от определенных проверяемых условий, таких как равенство, нулевое значение, отрицательное значение, больше чем и т. п.

В состав микропроцессора обычно входит также специальный регистр, называемый указателем стека (SP — Stack Pointer). Он содержит адрес ячейки, находящейся

[1] Или команда условного перехода, при которой заполнение счетчика команд зависит от результата предшествующей условному переходу операции. *(Прим. ред.)*

на дне свободного пространства стека. Стек—это особым образом организованный участок оперативной памяти для временного хранения содержимого внутренних регистров микропроцессора[1]. Такая память необходима в том случае, когда нужно прекратить выполнение реализуемой последовательности команд и вернуться к ней позже, например, для немедленного выполнения специальной подпрограммы или в результате прерывания программы. Текущее состояние счетчика команд и в случае прерывания состояние других регистров должно быть сохранено, с тем чтобы микропроцессор мог к ним вернуться и продолжить свою работу, начиная с момента, на котором она была прервана. Заметим, что если указателем стека служит 8-разрядный регистр, то стек ограничен 256 ячейками, размещенными на первой странице. Данные от микропроцессора поступают в верхнюю часть стековой памяти, по восемь разрядов одновременно, и содержимое указателя стека при этом уменьшается на единицу, чтобы всегда указывать на дно свободного пространства стека.

Когда данные выбираются (считываются) из стека (выталкиваются из магазина), содержимое указателя стека увеличивается на единицу с каждым выбранным байтом. Такое устройство памяти, кроме того, называют стеком типа LIFO (Last-In, First-Out) или «последним вошел, первым вышел».

В состав микропроцессора могут входить также индексные регистры, дающие возможность пользоваться в программах индексной адресацией. Индексный регистр используется для вычисления фактического адреса операнда в памяти в процессе выполнения команды путем суммирования содержимого индексного регистра с другим числом, содержащимся в самом адресе команды.

Иногда в состав микропроцессора входит также регистр вектора прерываний. Он загружается данными, которые комбинируются с кодом устройства, вызывающего прерывание (например, порт ввода), и формируют тем самым адрес памяти, указывающий на местоположение в памяти программы обработки прерываний для этого устройства. Например, входной порт может сообщить

---

[1] Так называемая стековая память или магазинная память. *(Прим. пер.)*

микропроцессору о своей готовности передавать данные, прервать основную программу и побудить микропроцессор перейти к выполнению другой стандартной подпрограммы, имеющей отношение к передаче данных. Когда передача данных будет закончена, микропроцессор может вернуться к выполнению основной программы.

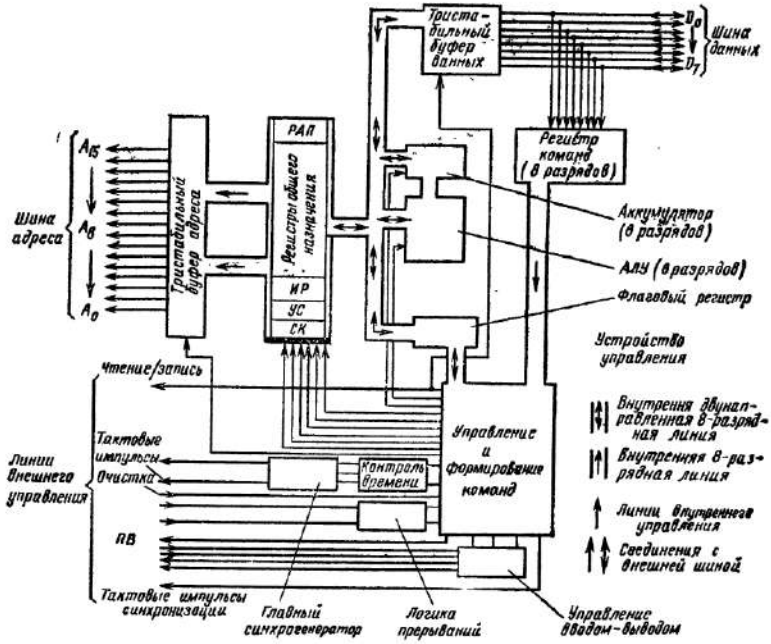


Рис. 9.2. Типовая архитектура микропроцессора:

УС — указатель стека; *СК* — счетчик команд; *ИР* — индексный регистр; *РАП* — регистр адреса памяти; *ПВ* — подключение выхода

Взаимосвязи между всеми рассмотренными элементами и устройствами, входящими в состав микропроцессора, указаны на рис. 9.2, представляющем собой схему типовой внутренней структуры микропроцессора.

Как неоднократно отмечалось, микро-ЭВМ не способна отличить данные, подлежащие обработке, от команд, которые необходимо выполнить. Если программист случайно поместит данные в область памяти, к которой микропроцессор обращается при поиске команды, произойдет ошибка, и, если команда действи-

тельно не будет найдена, микропроцессор остановится. Если же содержимое ячейки памяти случайно окажется командой, она будет выполнена, однако результат будет не тем, который ожидается программистом. Чтобы не допустить подобных ошибок, программист после написания программы должен внимательно, шаг за шагом, проследить за ее прохождением. Этот процесс называется отладкой программы.

Что подразумевается под прерыванием?

Сигналы прерывания предупреждают микропроцессор о том, что произошло некое внешнее событие, требующее с его стороны соответствующей реакции. Предположим, что из внешнего источника в микро-ЭВМ передаются в последовательной форме данные. Каждые восемь двоичных разрядов должны последовательно, по одному, передаваться в порт ввода, после чего микропроцессор считает их в параллельной форме. Естественно, что при этом между каждым поступлением нового байта данных (восемь двоичных разрядов) будет происходить задержка в их обработке. В течение каждого такого пропуска микропроцессор непрерывно следит за состоянием входного порта, выясняя, поступил ли новый байт. Это приводит к непроизводительной потере времени, так как микропроцессору приходится бездействовать в ожидании каждого байта данных.

Механизм прерываний дает возможность микропроцессору использовать свободные промежутки времени для выполнения другой части программы. Когда очередной байт данных готов для обработки, прерывание информирует об этом микропроцессор, и он выполняет ряд действий по обслуживанию прерываний, что позволяет выполнить некоторую обработку данных, прежде чем микропроцессор вернется к прерванной части программы.

Несколько портов ввода и вывода могут послать запрос на прерывание в одно и то же время. В таком случае механизм векторных прерываний определяет, какое из устройств требует обслуживания в первую очередь.

Прерывания используются также при обработке данных в масштабе реального времени, когда данные для ввода в микро-ЭВМ вырабатываются так быстро, что информация о том, что они готовы, должна управлять их

вводом в микро-ЭВМ. В противном случае, при задержке их ввода, данные могут быть утеряны.

Другие типы прерываний используются для того, чтобы вовремя подать сигнал о возникших ненормальных или угрожающих нормальной работе условиях, таких как угроза отказа питания или неисправности некоторых подсистем. Эти типы прерываний являются немаскируемыми (non-maskable), что означает, что они немедленно выполняются по окончании выполнения текущей команды программы[1].

Что такое синхрогенератор?

Синхрогенератор[2] (clock) вырабатывает непрерывную последовательность тактовых импульсов с регулярными интервалами. Один цикл в последовательности тактовых импульсов часто называется Т-состоянием (рис. 9.3, *а*).

Схему генератора импульсов можно получить довольно просто с помощью инверторов. На рис. 9.3, *б* приведена структурная схема такого генератора импульсов.

Если в точке *a* положительный потенциал U_+ (логическая 1), то в точке *b* потенциал равен 0 (логический 0), следовательно, в точке *c* появится положительный потенциал U_+, и, наконец, в точке *d* будет нулевой потенциал. Это приведет к тому, что в точке *a* потенциал сменится на 0, в точке *b* на U_+, в точке *c* на 0, в точке *d* на 1 и т. д. В результате потенциал U_+ будет перемещаться по кругу с частотой, определяемой задержками переключений инверторов. Любое нечетное число инвенторов в такой схеме будет генерировать импульсы

---

[1] Во многих микропроцессорах для обработки одновременно поступающих запросов на прерывание они упорядочиваются по приоритетам. Запросы с менее высоким приоритетом блокируются запросами, у которых он выше. Наивысший уровень приоритета запроса сравнивается с текущим приоритетом процессора. Если уровень приоритета прерывания выше уровня приоритета процессора, процессор воспринимает прерывание и начинает его обработку. В противном случае он продолжает выполнение программы. Механизмы приоритетов могут реализоваться как аппаратным, так и программным способом и значительно различаются у разных типов микропроцессоров. *(Прим. ред.)*

[2] Тактовый генератор, или генератор тактовых импульсов. *(Прим. ред.)*

аналогично, и для того, чтобы получить реальную схему генератора (синхрогенератора), следует понизить частоту следования импульсов до требуемого значения. Это можно сделать с помощью конденсатора (рис. 9.3, *в*), который не допустит мгновенного изменения состояния соединенного с ним входа инвертора. Конденсатор сначала должен зарядиться или разрядиться. Поэтому тре-

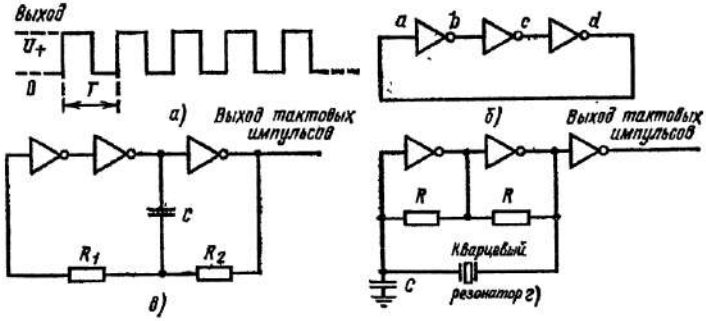


Рис. 9.3. Синхрогенератор:

а — тактовые импульсы; *б* — структурная схема генератора импульсов; *в* — принципиальная электрическая схема генератора импульсов; *г* — схема синхрогенератора с кварцевым резонатором

буемая частота генератора может быть определена подбором значений сопротивления R и емкости C цепей заряда и разряда. Если $R_1=R_2=R$, то частота, в герцах, $F=0{,}599/(RC)$, где R измеряется в омах, а C — в фарадах.

Если от генератора требуется высокая стабильность частоты следования импульсов, как при синхронизации микропроцессора, работающего с частотой, близкой к максимально возможной, в схему необходимо добавить кварцевый разонатор (рис. 9.3, *г*).

Как управляет работой микропроцессора синхрогенератор?

Каждую команду микропроцессора можно разбить на ряд основных операций, таких как выборка из памяти и запись в нее. Каждая из этих основных операций, называемых машинными циклами, занимает несколько циклов тактовых импульсов и состоит из двух основных частей: фазы выборки и исполнительной фазы. Как

команда разбивается на машинные циклы, так и машинные циклы можно разбить на микрооперации, такие как отпирание выхода регистра счетчика команд, посылка сигнала ADD в арифметическо-логическое устройство и т. п.

Далеко не все, а только некоторые команды требуют для своей реализации одного машинного цикла и выполняются на протяжении одной исполнительной фазы машинного цикла. Например, такой командой является CLEAR carry flag (ОЧИСТИТЬ флаг переноса), временна́я диаграмма которой приведена на рис. 9.4.

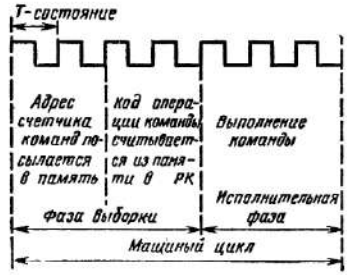


Рис. 9.4. Временная диаграмма для команды, требующей только одного машинного цикла:
РК — регистр команд

Команда, код операции и соответствующий адрес которой занимает более одной ячейки памяти, требует для своей реализации более одного машинного цикла, прежде чем завершится ее выборка. Микропроцессор бездействует на протяжении этих фаз выборки машинных циклов, либо фазы выборки пропускаются, пока вся команда не будет выбрана, после чего машинная команда выполняется в течение последующих машинных циклов. Передача всех данных происходит во время фаз выборки машинных циклов. На рис. 9.5 приведена временна́я диаграмма выборки и исполнения команды LOAD 102 (ЗАГРУЗИТЬ данные в ячейку 102). На рис. 9.6—9.9 представлены разбиения каждого из четырех машинных циклов.

В каждом машинном цикле требуется строго синхронизированная во времени последовательность управляющих сигналов для активирования необходимых путей прохождения данных в строго определенные моменты времени и выполнения требуемых микроопераций. Эти управляющие сигналы могут вырабатываться устройством управления дешифратором команд из операционного кода с использованием логических аппаратных средств и генераторов последовательностей сигналов,

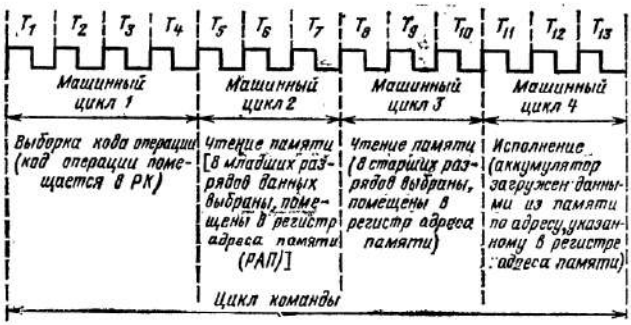


Рис. 9.5. Временная диаграмма для команды, требующей четырех машинных циклов

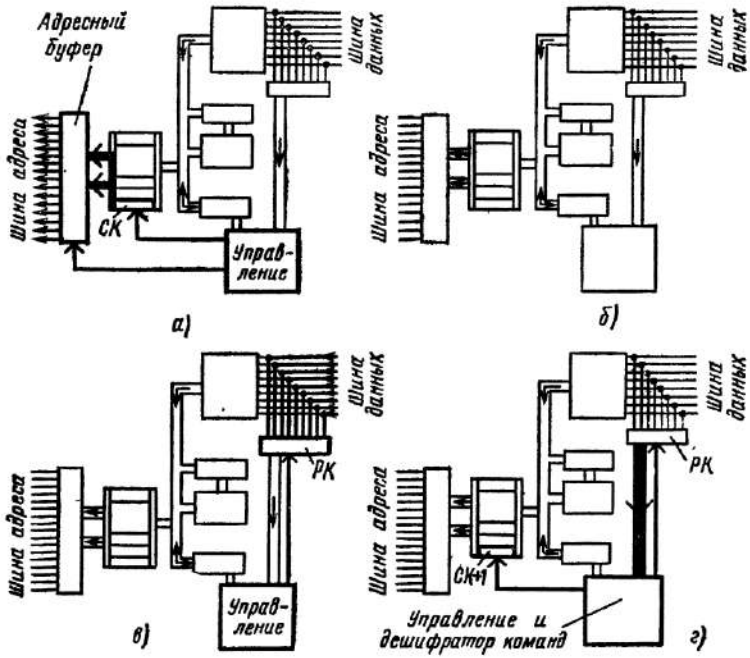


Рис. 9.6. Реализация первого машинного цикла:

а — цикл T_1 (счетчик команд→буфер адреса→адресная шина→память); *б* — цикл T_2 (никаких изменений по сравнению с T_1; предоставлено время для ответа памяти); *в* — цикл T_3 (данные в шине данных→регистр команд); *г* — цикл T_4 (операционный код команды из регистра команд→дешифратор; приращение содержимого счетчика команд)

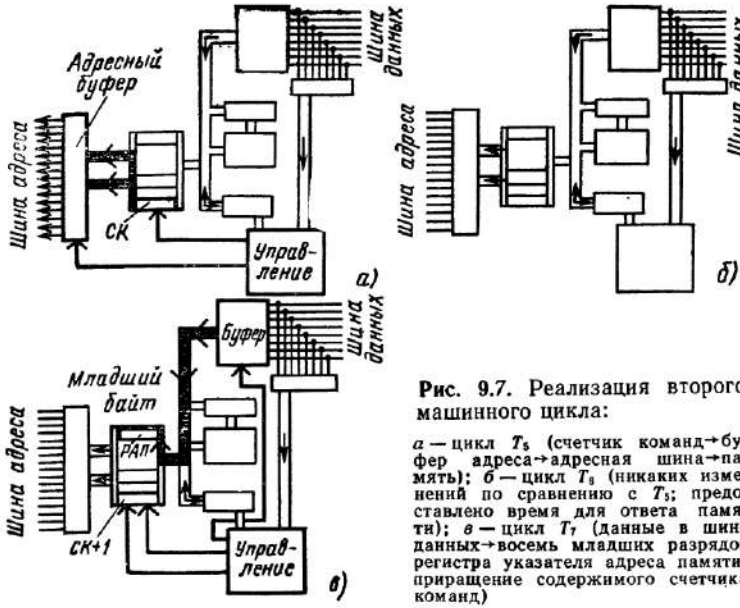


Рис. 9.7. Реализация второго машинного цикла:

а — цикл T_5 (счетчик команд→буфер адреса→адресная шина→память); *б* — цикл T_8 (никаких изменений по сравнению с T_5; предоставлено время для ответа памяти); *в* — цикл T_7 (данные в шине данных→восемь младших разрядов регистра указателя адреса памяти; приращение содержимого счетчика команд)

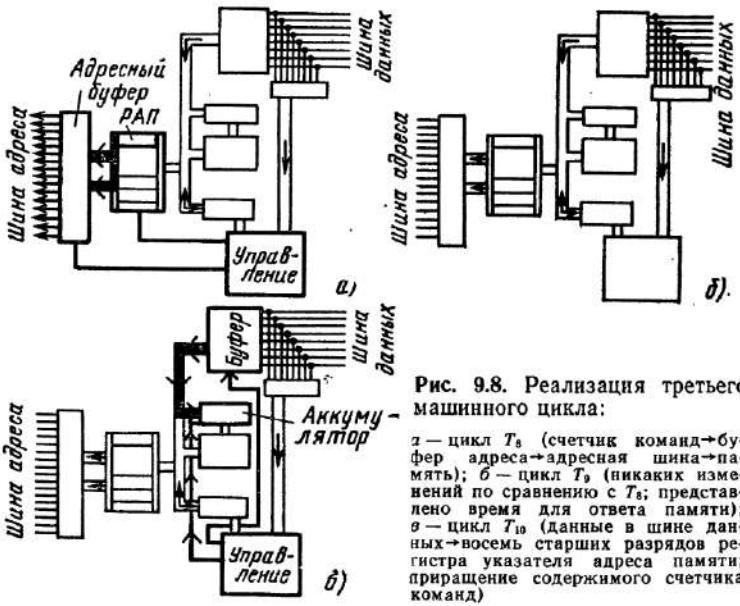


Рис. 9.8. Реализация третьего машинного цикла:

а — цикл T_8 (счетчик команд→буфер адреса→адресная шина→память); *б* — цикл T_9 (никаких изменений по сравнению с T_8; представлено время для ответа памяти); *в* — цикл T_{10} (данные в шине данных→восемь старших разрядов регистра указателя адреса памяти; приращение содержимого счетчика команд)

построенных на базе триггеров и комбинационных элементов. Однако для этого потребуется слишком большой расход аппаратуры. К меньшому расходу оборудования приведет схема, построенная на основе ПЗУ, информация в котором (единицы) используется для формирования управляющих сигналов. Можно было бы

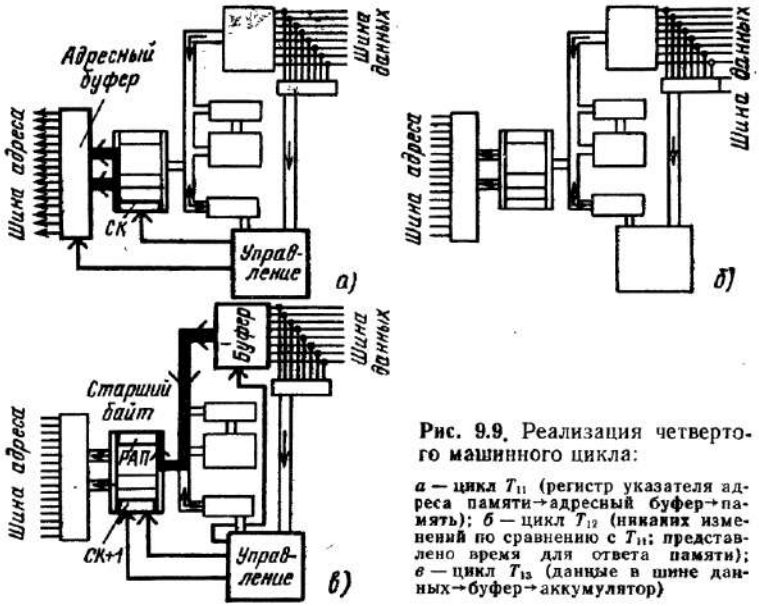


Рис. 9.9. Реализация четвертого машинного цикла:

а — цикл T_{11} (регистр указателя адреса памяти→адресный буфер→память); *б* — цикл T_{12} (никаких изменений по сравнению с T_{11}; представлено время для ответа памяти); *в* — цикл T_{13} (данные в шине данных→буфер→аккумулятор)

хранить в ПЗУ целую последовательность нескольких «слов-микрокоманд» для того, чтобы каждая команда и содержимое счетчика могли в дальнейшем использоваться для считывания в линию управления каждого управляющего слова в порядке расположения.

Различные команды могли бы реализоваться при наличии отдельных ПЗУ для каждой команды и условии использования сигналов с выходов дешифратора команд для отпирания (считывания) только требуемого ПЗУ.

Однако для набора всех необходимых команд в этом случае потребуется довольно много отдельных ПЗУ. В 1952 г. Уилкс и Стринжер предложили воспользоваться микропрограммами, содержащимися в ПЗУ, для

обеспечения выполнения отдельных команд, а также для формирования управляющих сигналов. Именно на этом принципе основана работа управляющего устройства микропроцессора. Дешифратор команд и управляющее устройство (control section) представляют собой «микропроцессор в микропроцессоре» специального назначения[1]. Дешифратор микропрограмм содержит уп-

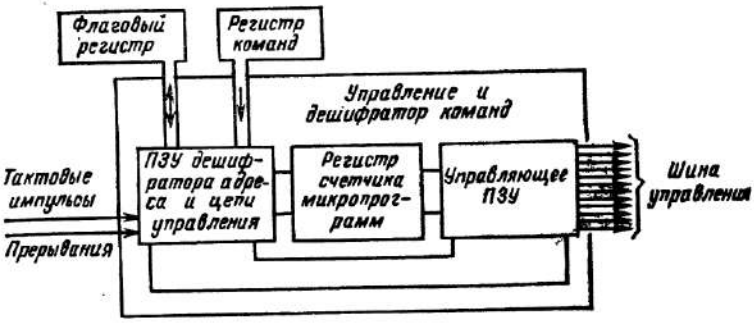


Рис. 9.10. Структурная схема микропрограммного управления

равляемое ПЗУ, запрограммированное при изготовлении, с хранящимися в нем микрокомандами для выполнения специального набора команд. Операционный код каждого используемого уровня машинного языка дешифрируется, в результате чего формируется начальный адрес в микропрограмме для данной команды. Часть кода операции машинного языка используется также для того, чтобы определить, где микропрограмма должна сделать переход к другой микропрограмме, требуемой для реализации данной команды.

Ветвление программы может зависеть также от состояний флагов во флаговом регистре. Использование микропрограмм позволяет резко сократить объем требуемого ПЗУ и всю аппаратуру поместить в управляющем устройстве внутри микропроцессора. Как только микропрограмма построена, микрокоманды считываются в нужном порядке и выполняют необходимые микрооперации. На рис. 9.10 приведена структурная схема такой системы.

---

[1] Микропрограммный автомат, или схему Уилкса. *(Прим. ред.)*

Что подразумевается под двухфазным синхрогенератором?

Некоторым микропроцессорам требуется внешний синхрогенератор. Он может быть двух типов: однофазный и двухфазный. Двухфазный означает, что синхрогенератор имеет два выхода, сигналы которых находятся в противофазах по отношению друг к другу. Други-

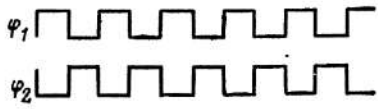


Рис. 9.11. Диаграмма выходных сигналов двухфазного синхрогенератора

ми словами, в любой момент времени сигналы на двух выходах имеют разные уровни напряжения. На рис. 9.11 приведены сигналы двух выходов синхрогенератора, находящихся в противофазах. Когда на одном из них имеется положительный потенциал U_+, на другом 0, и наоборот.

Часто сигналы центрального (главного) синхрогенератора используются для генерации тактовых импульсов меньшей частоты, применяющихся в других, менее быстродействующих устройствах микро-ЭВМ, таких, например как память. Более быстродействующие устройства микро-ЭВМ, таким образом, способны выполнять несколько операций в каждом цикле работы устройств с меньшим быстродействием.

Какие еще встречаются схемы управления?

Среди дискретных устройств, используемых в схемах управления, можно встретить мультиплексор. Это переключатель с несколькими входами и одним выходом, который может подсоединить любой из входов (только один) к своему выходу или не подсоединять ни одного в зависимости от поступающих на него управляющих сигналов. Мультиплексоры используются для соединения с информационными каналами устройств микро-ЭВМ, не имеющих выходов с тремя состояниями. Это предотвращает ошибочное появление на одной линии шины одновременно более одного сообщения.

Схему однонаправленного мультиплексора можно построить из элементов И и ИЛИ, либо применив бу-

фера с тремя состояниями выходов (тристабильные буфера), описанные в гл. 8. На рис. 9.12 приведена схема из таких буферов.

Схема двунаправленного мультиплексора строится таким образом, чтобы поток данных мог продвигаться

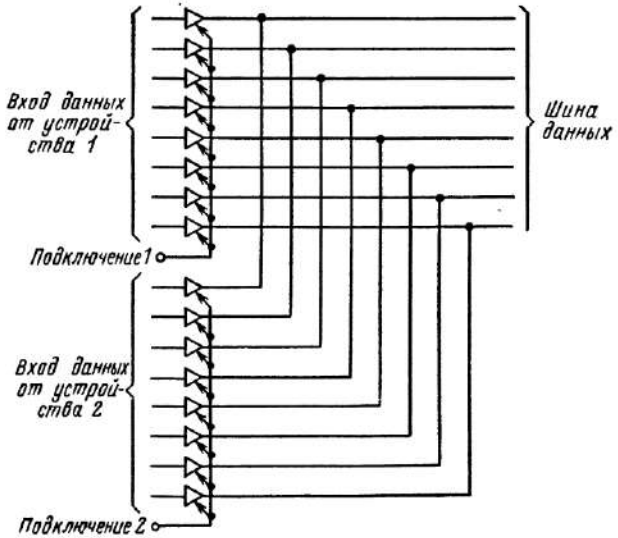


Рис. 9.12. Схема подсоединения выходных двух устройств к шине данных с помощью тристабильных буферов

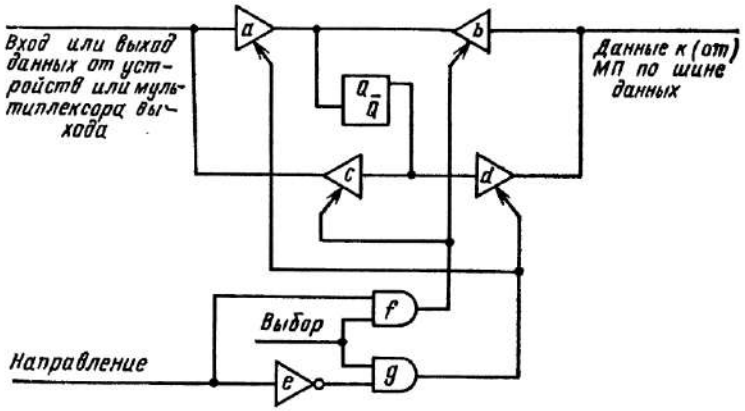


Рис. 9.13. Схема двунаправленного мультиплексора

в обоих направлениях, но одновременно только по одному из них. На рис. 9.13 приведена схема такого мультиплексора.

Когда управляющий направлением сигнал (на входе НАПРАВЛЕНИЕ) равен 0, элемент *f* блокирован, поэтому выходной сигнал этого элемента также равен 0, и тристабильные буфера *c* и *b* блокированы. Однако элемент *g* не заблокирован благодаря инвертору *e*. Поэтому, если сигнал выбора мультиплексора (на входе ВЫБОР) равен U_+, то сигнал на выходе элемента *g* будет также равен U_+, и тристабильные буфера *a* и *d* будут открыты. Таким образом, данные могут считываться слева через буфер *a* в регистр мультиплексора и далее через буфер *d* поступать в микропроцессор.

Аналогично, если управляющий направлением сигнал равен U_+, данные могут передаваться из микропроцессора через буфер *b* в регистр мультиплексора и из него далее через буфер *c*.

Когда сигнал выбора мультиплексора равен 0, то все четыре тристабильных буфера будут переведены в состояние высокого сопротивления (блокированы), и передача данных в каком-либо направлении станет невозможной.

ГЛАВА ДЕСЯТАЯ

Ввод-вывод

Как в микропроцессоре осуществляется ввод и вывод информации?

Простейший способ отображения двоичной информации состоит в использовании восьми светоизлучающих диодов (LED — Light-Emitting Diode), или светодиодов, подключенных к шине данных.

Несколько сложнее устроены семисегментные светодиодные индикаторы (рис. 10.1, *а*). Они предназначены для отображения выходной информации в виде десятичных цифр путем включения определенных комбинаций светящихся сегментов (рис. 10.1, *б*). Индикатор получает управляющие сигналы, поступающие на семь входов (*a*, *b*, *c*, *d*, *e*, *f*, *g*), от специального формирователя (рис. 10.1, *в*), в состав которого входит преобразователь двоичных сигналов из шины данных в сигналы специального кода, соответствующего правильным комбинациям высвечиваемых сегментов.

Для ввода информации в двоичной форме в микроЭВМ, если не иметь в виду случай, когда на каждый двоичный разряд отводится своя клавиша, требуется

специальный двоичный код. Десятичные числа могли бы передаваться их двоичными эквивалентами. Однако для букв и знаков пунктуации необходим особый код. Наиболее распространенным кодом для микро-ЭВМ является американский стандартный код для обмена ин-

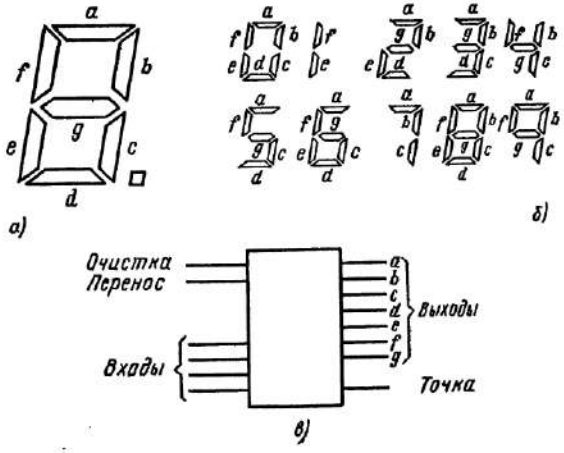


Рис. 10.1. Семисегментный светодиодный индикатор:

а — семь сегментов (*a*, *b*, *c*, *d*, *e*, *f*, *g*) и еще один для десятичной точки (запятой); *б* — конфигурации десятичных цифр; *в* — входы и выходы формирователя

формацией ASCII (American Standard Code for Information Interchange). Код ASCII служит для представления буквенно-цифровой информации, знаков пунктуации и специальных (в том числе управляющих) символов:

Символ	Семиразрядный код ASCII	Символ	Семиразрядный код ASCII
A	100 0001	a	110 0001
B	100 0010	b	110 0010
C	100 0011	c	110 0011
D	100 0100	d	110 0100
E	100 0101	e	110 0101
F	100 0110	f	110 0110
G	100 0111	g	110 0111
H	100 1000	h	110 1000
I	100 1001	i	110 1001

Символ	Семиразрядный код ASCII	Символ	Семиразрядный код ASCII
J	100 1010	j	110 1010
K	100 1011	k	110 1011
L	100 1100	l	110 1100
M	100 1101	m	110 1101
N	100 1110	n	110 1110
O	100 1111	o	110 1111
P	101 0000	p	111 0000
Q	101 0001	q	111 0001
R	101 0010	r	111 0010
S	101 0011	s	111 0011
T	101 0100	t	111 0100
U	101 0101	u	111 0101
V	101 0110	v	111 0110
W	101 0111	w	111 0111
X	101 1000	x	111 1000
Y	101 1001	y	111 1001
Z	101 1010	z	111 1010
0	011 0000	!	010 0001
1	011 0001	"	010 0010
2	011 0010	#	010 0011
3	011 0011	%	010 0101
4	011 0100	&	010 0110
5	011 0101	'	011 1010
6	011 0110	∨	011 1100
7	011 0111	∧	011 1110
8	011 1000	?	011 1111
9	011 1001	@	100 0000
Пробел	000 0000	ℒ	101 1100
(	010 1000	↑	101 1110
+	010 1011	—	101 1111
}	101 1101	Пробел	010 0000
{	101 1011		
$	010 0100		
*	010 1010		
)	010 1001		
;	011 1011		
,	010 1100		
.	010 1101		
=	011 1101		
.	010 1110		

Информация вводится в микро-ЭВМ нажатием кнопки клавишного пульта с соответствующим знаком. Преобразование вводимой информации в сигналы кода ASCII, передаваемые в шину данных, осуществляется либо с помощью соответствующей схемы, либо с помощью специальной подпрограммы, являющейся частью

управляющей программы (программа, называемая монитор). Монитор при вводе информации сдвигает на экране дисплея каждую строку вверх на одну позицию, когда строка заполнена, и перемещает курсор к началу новой строки. Курсор — это специальный символ, указывающий место в строке, куда будет помещен очередной знак вводимой информации. Он обычно имеет вид небольшого квадрата или черточки, помещенных чуть ниже уровня строки. Каждый знак (цифра, буква или специальный символ) образуется комбинацией высвечиваемых точек матрицы. Наиболее распространенные системы знаков имеют матрицы размерности либо 7×5, либо 9×7.

Первые генераторы знаков воспроизводили только наборы знаков в матрице размерности 7×5 из-за ограничений на объем памяти, необходимой для работы генераторов. Каждая точка матрицы должна была храниться в ПЗУ как 1 в случае ее высвечивания на экране дисплея или как 0 в случае затемнения. Однако современные ПЗУ, выполненные по МОП-технологии, позволяют использовать память на одной микросхеме гораздо большего, чем ранее, объема. Поэтому сейчас широкое распространение получили наборы из 128 знаков с размерностью матрицы 9×7 (рис. 10.2), так же как и генераторы знаков на одной микросхеме (8 К двоичных разрядов). При использовании в качестве дисплеев телевизионных экранов (с горизонтальной разверткой) и печатающих устройств (принтеров) каждая строка знаков представлена сигналами на параллельных выходах генератора знаков, откуда они затем пересылаются в сдвиговый регистр и используются при последовательном вводе для управления лучом телевизионной трубки или механизмом печатающего устройства (черная точка при печати соответствует 1).

Существует способ отображения знаков на экране дисплея, при котором в функции программного обеспечения входит определение знака, предназначенного для печати, и места его расположения на экране. При этом на экране размещается определенное число строк, а в каждой строке — выделяется определенное число позиций для знаков. Начало и конец каждой строки не используются, поскольку бытовые телевизоры, как правило, имеют слишком сильно развернутый экран, и края строк исказятся, если не предусмотреть пустых полей

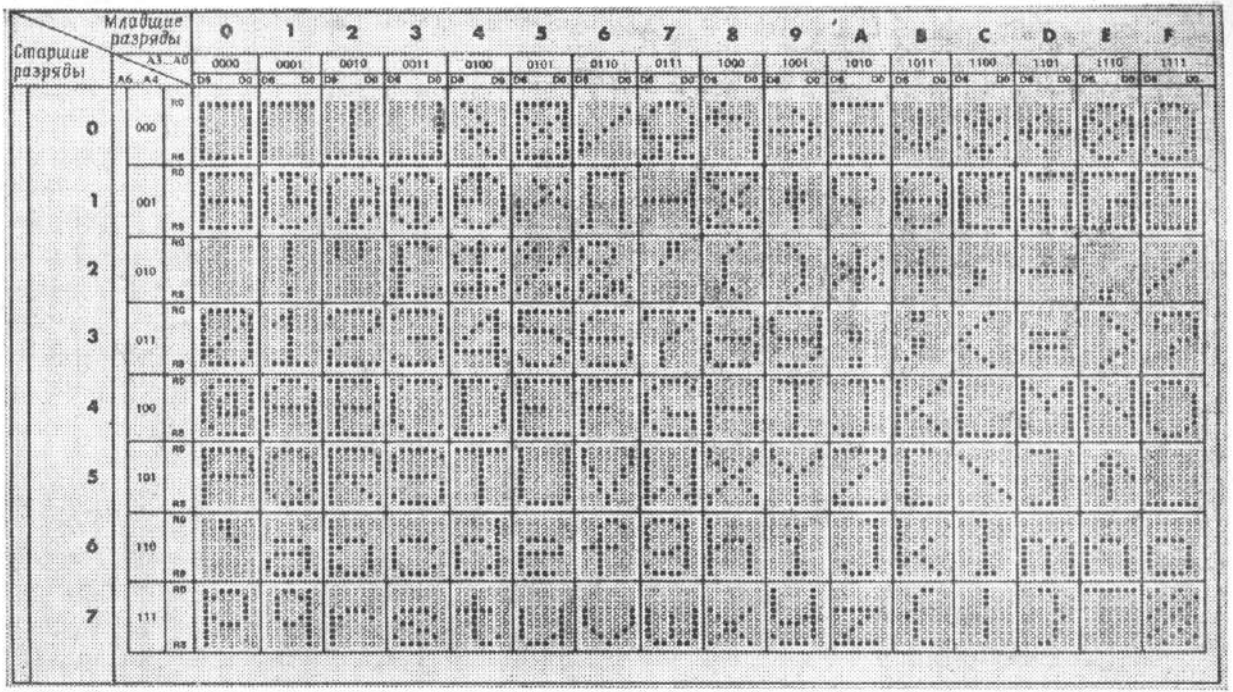


Рис. 10.2. Комплект конфигураций знаков

по сторонам. В некоторых телевизорах этого недостатка можно избежать регулированием размера строк по горизонтали. Позиции каждого символа на экране соответствует определенное место в памяти. Код AS CII для выводимого на экран знака должен передаваться в соответствующее место памяти и затем в генератор знаков в заданное время в соответствии с принципом адресации внешних устройств (в данном случае дисплея) как ячеек памяти.

Таким способом можно организовать адресацию любого из внешних устройств, а области памяти могут отводиться определенным устройствам ввода-вывода. Однако, если в микропроцессоре предусмотрены специальные команды и средства запроса ввода-вывода, система адресации устройств ввода-вывода как ячеек памяти становится ненужной.

Как микропроцессор осуществляет связь с внешними устройствами?

В предыдущих главах упоминалось об особенностях сопряжения внешних устройств, ориентированных на последовательный ввод и вывод информации, с микро-ЭВМ с помощью соответствующих адаптеров сопряжения с внешними устройствами. Адаптеры содержат сдвиговые регистры, а также буфера или специальные преобразователи, обеспечивающие согласование уровней (по напряжению и мощности) для связи схем из различных логических элементов, например МОП-транзисторов, элементов ТТЛ или механических реле и т. п.

Адаптеры сопряжения часто сами могут программироваться. В микро-ЭВМ, как правило, используется асинхронный обмен информацией между внешними устройствами и микропроцессором, иногда называемый еще хэндшейком ввода-вывода. Если бы двоичные разряды для каждого знака кода AS CII выдавались в виде непрерывной цепочки, то их обработку микро-ЭВМ пришлось бы синхронизировать с работой передающего устройства, чтобы отслеживался момент, когда один символ заканчивается, а следующий начинается. Для преодоления этой сложной проблемы согласования во времени при асинхронной передаче используются стартовый и стоповый двоичные разряды в начале и конце каждого символа. Они согласуют работу принимающе-

го и передающего устройств, заставляя первый останавливаться после приема каждого знака и ожидать начала поступления следующего. Естественной представляется такая организация посылки 8-разрядного слова, при которой использовался бы один дополнительный стартовый двоичный разряд и один дополнительный стоповый разряд, составляющие вместе со словом десять двоичных разрядов. Если бы скорость передачи данных составляла 200 разрядов/с, то скорость в бодах была бы равна $8 \times 20 = 160$ бод, так как в течение каждой секунды проходило бы также 20 стартовых и 20 стоповых разрядов, которые не принимаются в расчет при определении скорости в бодах.

Адаптер последовательного интерфейса асинхронного обмена (ACIA) может преобразовывать получаемые последовательные данные, сигналы стартовых и стоповых разрядов, а также разрядов обнаружения ошибки (если таковые посылаются) в выходные параллельные данные, контролировать передачу свободных от ошибок данных и генерировать сигналы прерывания. Он также может преобразовывать параллельные данные, принятые по внутренней информационной шине, в последовательные и генерировать и добавлять особые сигналы стартовых (обычно один) и стоповых (обычно один или два) разрядов, а также любые сигналы проверки ошибки (например, проверки на четность), если потребуется.

Универсальный асинхронный приемопередатчик (UART) может также передавать и принимать асинхронные последовательные данные во всех принятых форматах слов.

Интерфейсы параллельного ввода-вывода могут также быть программируемыми и могут содержать несколько параллельных регистров, действующих как выходные порты, а также устройства логики для прерываний.

Во время пересылки данных от одного устройства к другому могут возникать ошибки, например, из-за электрических помех в шине. Поэтому часто применяется какой-нибудь вид проверки соответствия полученных данных переданным.

Один из таких видов проверки известен как контроль суммы. Предположим, необходимо передать несколько последовательных 8-разрядных слов. Можно разбить их на равные группы, сложить слова в каждой

группе друг с другом и посылать сигналы младших восьми разрядов суммы каждой группы вслед за группой в качестве сигналов проверки суммы. В действительности посылается обратный код проверки суммы. При получении данных вновь подсчитывается сумма, и значения младших восьми разрядов, содержащихся в аккумуляторе, прибавляются к полученному значению, чтобы выяснить, не возникли ли какие-либо ошибки при передаче. Полученный ответ всегда должен иметь следующий вид: $1111\ 1111_2$, так как для каждой цифры $Q_i + \overline{Q}_i = 1$.

Обнаружить ошибку в единичном байте можно с помощью определенных кодов. Таким, например, является код контроля четности. К каждому посылаемому символу добавляется вспомогательный контрольный двоичный разряд четности, содержимое которого выбирается таким образом, чтобы общее число единиц, включая

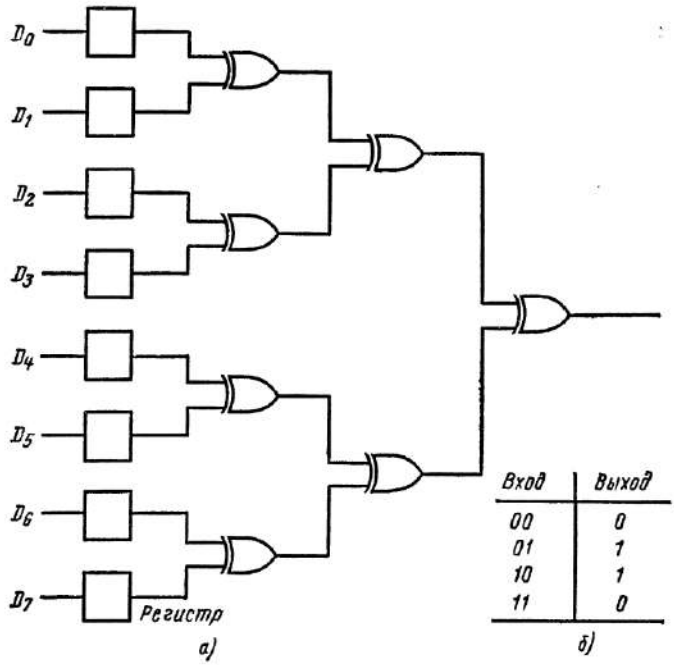


Рис. 10.3. Схема контроля четности:

а — схема; *б* — таблица истинности элемента ИСКЛЮЧАЮЩЕЕ ИЛИ

контрольный разряд четности, в каждом слове было четным. На рис. 10.3, *а* приведена схема контроля четности для четного числа входов. В этой схеме единичный сигнал на выходе указывает на наличие нечетного числа единиц в слове $D_0D_1D_2D_3D_4D_5D_6D_7$ и может использоваться при посылке сигнала корректировки. Та же схема может быть использована для определения ошибки в передаче, так как сформирует единичный сигнал на выходе, если слово данных содержит нечетное число единиц. В правильности работы схемы нетрудно убедиться путем подачи на входы нескольких различных слов. Для удобства на рис. 10.3, *б* приведена таблица истинности для элемента ИСКЛЮЧАЮЩЕЕ ИЛИ. При применении подобных схем производится проверка кода передаваемого знака. Если общее число единиц нечетное, то посылка не принимается, и выдается сообщение об ошибке. Иногда применяется контроль по нечетности, т. е. передача организуется так, чтобы общее число единиц в каждом коде было нечетным.

Микро-ЭВМ настолько надежны по сравнению с прежними вычислительными машинами, что проверяющие ошибки коды не применяются для внутренней передачи данных, а используются только для связи с внешними устройствами.

Иногда применяется синхронная передача данных между микропроцессором и внешними устройствами, такими, например, как устройство управления флоппидиском. Здесь не используются стартовые и стоповые разряды, а периферийные устройства синхронизируются с микропроцессором.

Применяется и универсальный синхронный/асинхронный приемопередатчик (USART) — устройство, способное преобразовывать параллельные данные фактически в любой требуемый формат последовательной передачи, синхронной или асинхронной.

ГЛАВА ОДИННАДЦАТАЯ

Краткое введение в технику программирования

Ранее упоминалось, что характерной особенностью микропроцессора, делающей его столь мощным вычислительным средством, является его способность следовать не только строго фиксированному порядку выполне-

ния команд, но и при определенных обстоятельствах осуществлять условные и безусловные переходы.

Как микропроцессор реализует переход в программе?

Организовать переходы в программе позволяет применение специальных команд такого типа, как:

BRANCH unconditional — ПЕРЕХОД безусловный. Эта команда устанавливает счетчик команд в состояние, соответствующее адресу в памяти, выпадающему из общей последовательности адресов. При этом очередная команда программы выбирается из памяти по новому адресу (другое название команды JUMP unconditional).

BRANCH conditional — ПЕРЕХОД условный. Если встречается особое условие, на которое есть ссылка в команде (обычно проверка установки определенного флага), счетчик команд при этом устанавливается так же, как описывалось в предыдущем случае. Очередная команда выбирается из памяти тоже по новому адресу (другое название этой команды JUMP conditional).

CALL subroutine — ВЫЗОВ подпрограммы. По этой команде содержимое счетчика команд получает приращение, а адрес следующей команды запоминается в стеке. Он служит адресом возврата из подпрограммы. Счетчик команд далее загружается адресом подпрограммы, и процедура продолжается.

CONDITIONAL CALL subroutine — УСЛОВНЫЙ ВЫЗОВ подпрограммы. (Вызов в зависимости от состояния соответствующего флага условия).

RETURN — ВОЗВРАТ. По этой команде осуществляется выборка адреса возврата, находящегося в стеке, и загрузка его в счетчик команд с тем, чтобы после возвращения из подпрограммы могла далее выполняться основная программа.

CONDITIONAL RETURN — УСЛОВНЫЙ ВОЗВРАТ (возврат в зависимости от состояния соответствующего флага условия).

Какие этапы необходимо выполнить при составлении сложной программы?

Основные этапы составления программы показаны на рис. 11.1. При программировании на языке высокого уровня трансляция и загрузка выполняются по специ-

альным программам. При отладке программ существенную помощь оказывает управляющая программа (монитор).

Прежде чем приступить к написанию программы, необходимо сформулировать основную идею построения схемы вычислений, называемой схемой алгоритма. Ранее

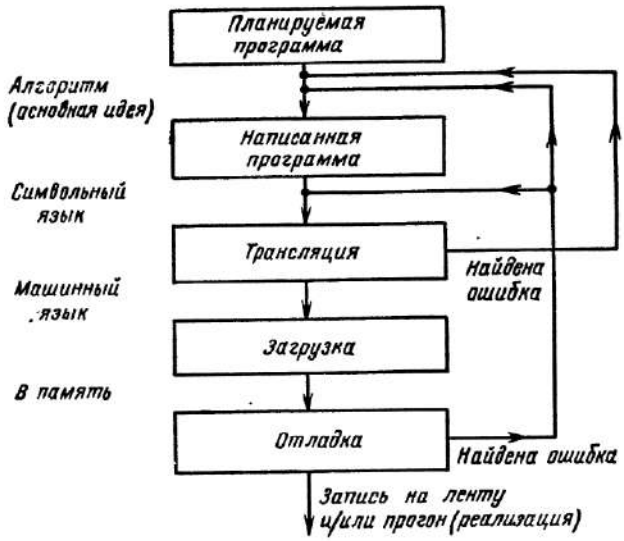


Рис. 11.1. Основные этапы составления программы

рассматривалась программа сложения двух чисел. Предположим, что теперь нужно сложить все целые числа в интервале от 1 до 200, т. е. получить сумму 1+2+3+... ...+199+200. Чтобы вычислить эту сумму непосредственно, потребуется очень большое число шагов в программе и записывать все это будет слишком утомительно. Можно, однако, составить программу с использованием команд перехода, что существенно сократит число шагов, необходимых для ее выполнения.

Основную идею алгоритма, реализуемого программой, можно было бы выразить следующим образом. «Необходимо получить результат, который назовем SUM (суммой). Обозначим его через S. Пусть имеется число N, равное, как и сумма S, 0. Найдем число S+N и назовем

его NEW SUM (новой суммой). Затем находим число N+1 и проверяем, меньше ли оно числа 201. Если да, то процесс вычислений продолжается. Если N+1=201, выполнение программы прекращается, а полученное значение S будем считать искомым результатом».

Алгоритм вычислений чаще легче понять, чем представить его специальной диаграммой, называемой схемой алгоритма[1]. Схема алгоритма сложения для рассматриваемой задачи приведена на рис. 11.2. Как следует из рисунка, схема алгоритма — это ориентированная сеть с вершинами различных типов.

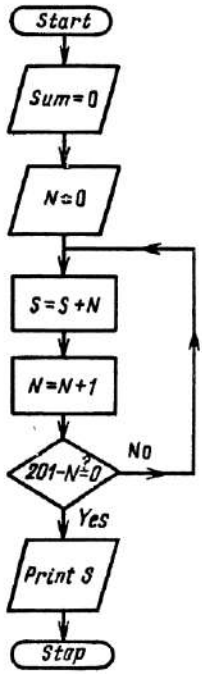


Рис. 11.2. Схема алгоритма сложения целых чисел от 1 до 200

Как превратить схему алгоритма в программу?

На языке БЕЙСИК рассматриваемая программа выглядела бы следующим образом:

```
10 LET S=0
20 LET N=0
30 LET S=S+N
40 LET N=N+1
50 IF N<201 GO TO 30
60 PRINT S
70 END
```

После прохождения программы выпечатывается ответ.

Просмотрим еще раз все упоминавшиеся ранее команды для воображаемого микропроцессора, описанного в тексте, после чего станет ясно, можно ли написать подобную программу для него в шестнадцатеричном коде:

Команда	Мнемокод	Символ операции
LOAD (ЗАГРУЗИТЬ) аккумулятор	LDA	M→A
ADD (СЛОЖИТЬ) без переноса	ADD	M+A→A
ADD (СЛОЖИТЬ) с переносом	ADC	M+A+CF→A

[1] Подобные диаграммы ранее назывались блок-схемами алгоритмов. Здесь и далее используется название по ГОСТ. (*Прим. пер.*)

Продолжение

Команда[1]	Мнемокод	Символ операции
STORE (ЗАПОМНИТЬ) содержимое аккумулятора	STA	$A \rightarrow M$
INCREMENT (УВЕЛИЧИТЬ) содержимое аккумулятора	INC	$A+1 \rightarrow A$
COMPLEMENT (ДОПОЛНЕНИЕ ДО 1) содержимого аккумулятора	COMP	$\overline{A} \rightarrow A$
AND (логическое И) содержимого памяти и аккумулятора	AND	$M \cdot A \rightarrow A$
OR (логическое ИЛИ) содержимого памяти и аккумулятора	OR	$M+A \rightarrow A$
XOR (ИСКЛЮЧАЮЩЕЕ ИЛИ) содержимого памяти и аккумулятора	XOR	$M \oplus A \rightarrow A$
CLEAR CARRY FLAG (ОЧИСТИТЬ ФЛАГ ПЕРЕНОСА)	CCF	$0 \rightarrow CF$
CLEAR ACCUMULATOR (ОЧИСТИТЬ АККУМУЛЯТОР)	CLA	$0 \rightarrow A$
SHIFT (СДВИГ) содержимого аккумулятора вправо	SAR	$0 \rightarrow 76543210 \rightarrow CF$
SHIFT (СДВИГ) содержимого аккумулятора влево	SAL	$CF \leftarrow 76543210 \leftarrow 0$
STOP (СТОП)	BRK	PC stops

где M — ячейка памяти, A — аккумулятор, CF — флаг переноса, PC — счетчик команд.

Отметим также ранее упоминавшиеся флаги: флаг переноса, флаг нуля, флаг переполнения, флаг знака или флаг отрицательности. Они будут выставляться (устанавливаться) или сбрасываться в соответствии с результатом каждой выполненной арифметической или логической операции. Например, для операции ADD можно рассмотреть следующие случаи установки флагов:

1) установить флаг переноса, если есть перенос из старшего (седьмого) разряда, и сбросить его, если переноса нет;

2) установить флаг нуля, если результат — нуль, и сбросить его в противном случае (если результат не равен нулю);

3) установить флаг переполнения, если значение седьмого разряда результата отличается от значения седь-

[1] У читателя не должно сложиться впечатление, что приведенный список наиболее употребительных команд — это все, чем располагает типовой микропроцессор. Например, список команд микропроцессора Intel 8080 включает в себя 111 наименований (см. [6]). *(Прим. пер.)*

мого разряда любого операнда, и сбросить флаг переполнения в противном случае;

4) установить флаг знака, если значением седьмого разряда результата служит единица (сигнал о том, что число отрицательное), и сбросить флаг знака, если значение седьмого разряда результата —нуль (сигнал о том, что число положительное).

При программировании для конкретного микропроцессора Вам следует внимательно изучить набор его команд, чтобы выяснить, какие в нем содержатся флаги и какие команды на какие из флагов влияют. Между разными типами микропроцессоров в этом смысле различия весьма существенны.

Информация о том, какие команды на какие флаги оказывают влияние, для рассматриваемого микропроцессора может быть представлена в следующем виде:

Команда	Мнекомод	Устанавливаемые флаги			
		C	Z	O	S
LOAD аккумулятор	LDA	·	·	·	·
ADD без переноса	ADD	*	*	*	*
ADD с переносом	ADC	*	*	*	*
STORE содержимое аккумулятора	STA	·	·	·	·
INCREMENT содержимое аккумулятора	INC	·	·	·	·
COMPLEMENT содержимое аккумулятора	COMP	·	·	·	·
AND памяти и аккумулятора	AND	·	*	·	*
OR памяти и аккумулятора	OR	·	*	·	*
XOR памяти и аккумулятора	XOR	·	*	·	*
CLEAR CARRY FLAG	CCF	0	·	·	·
CLEAR ACCUMULATOR	CLA	0	1	0	0
SHIFT правый	SAR	*	*	·	0
SHIFT левый	SAL	*	*	·	*
STOP	BRK	·	·	·	·

Здесь используются следующие обозначения: · — на флаг влияния не оказывается; 0 — флаг устанавливается в 0; 1—флаг устанавливается в 1; * — флаг устанавливается или сбрасывается в зависимости от результата операции; C — флаг переноса; Z—флаг нуля; O—флаг переполнения; S — флаг знака.

К этому набору команд добавим еще три команды, ни одна из которых на состояние флагов не влияет: BRANCH UNCONDITIONAL (ПЕРЕХОД БЕЗУСЛОВНЫЙ), BRANCH ON ZERO (ПЕРЕХОД ПО НУЛЮ) и

BRANCH ON NOT ZERO (ПЕРЕХОД ПО НЕ НУЛЮ). Эти команды записывают в счетчик команд определенные значения либо безусловно, либо при условии наличия нулевого или ненулевого значения соответственно в зависимости от того, установлен флаг нуля или нет.

Мнемокоды и символическое представление операций для этих команд выглядит следующим образом:

Команда	Мнемокод	Символическое описание
BRANCH UNCONDITIONAL	BR	PC→ * * * *
BRANCH ON ZERO	BR *Z*	Если флаг нуля установлен PC→ * * * *
BRANCH ON NOT ZERO	BR *NZ*	Если флаг нуля не установлен PC→ * * * *

Здесь ** ** — адрес, указанный после кода операции, а PC — обозначение счетчика команд.

Посмотрим, можно ли воплотить приведенную на рис. 11.2 схему алгоритма сложения первых 200 чисел в программу. При одинарной точности арифметических вычислений это было бы сделать трудно, так как ответ (20100_{10}) больше максимального двоичного числа без знака, которое можно разместить в 8-разрядном регистре. Поскольку эта программа интересует нас только в качестве примера того, как можно перейти от схемы алгоритма к распечатке (листингу — listing) машинных команд в шестнадцатеричном коде, чтобы затем проанализировать, как эти команды реализуются в микро-ЭВМ, сохраним этот простой пример выполнения программы арифметического сложения, но складывать будем не все 200 чисел, а только первые 20. В этом случае полученный ответ уместится в 8-разрядном регистре как число без знака (число без знака означает, что можно не заботиться об использовании седьмого двоичного разряда для указания на то, положительное или отрицательное число получено в результате, т. е. содержимое

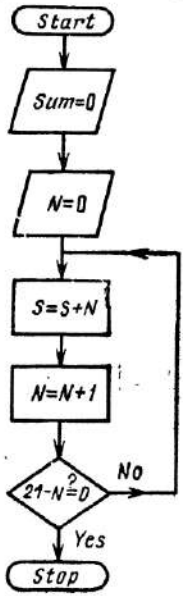


Рис. 11.3. Схема алгоритма сложения первых двадцати чисел

седьмого разряда может служить частью модуля числа). Схема алгоритма сложения первых 20 целых чисел приведена на рис. 11.3.

Представленная списком команд, подобных тем, которые использовались в гл. 3 при описании образной модели «стеллажной» микро-ЭВМ, программа будет иметь следующий вид:

Адрес	Команда	Аргумент	Комментарий
000	LOAD	101	Вычислить N
001	ADD без переноса	102	Новое S=S+N
002	STORE	102	Запомнить новое S
003	LOAD	101	Вычислить N
004	INC аккумулятор	—	Новое N=N+1
005	STORE	101	Запомнить новое N
006	COMP аккумулятора	—	Дополнительный код нового N
007	INC аккумулятор	—	
008	ADD без переноса	100	21 — новое N
009	BRANCH ON NOT ZERO	000	Цикл до выполнения условия
010	STOP	—	
100	Эта ячейка загружена числом 21		
101	Эта ячейка загружена начальным значением N(N=0)		
102	Эта ячейка загружена начальным значением S(S=0)		

Команды о печати результата решения задачи нет. После прохождения программы ответом будет содержимое ячейки 102 (рис. 11.4).

Эта программа на языке ассемблера выглядела бы следующим образом:

Метка	Мнемокод	Аргумент
START	LDA	101
	ADD	102
	STA	102
	LDA	101
	INC	
	STA	101
	COMP	
	INC	
	ADD	100
	BP NZ,	START
	BRK	

Адреса ячеек 100, 101 и 102 имеют отношение только к модели воображаемой «стеллажной» микро-ЭВМ. Что же касается любого реального микропроцессора, то для его программирования пользуются символическими адре-

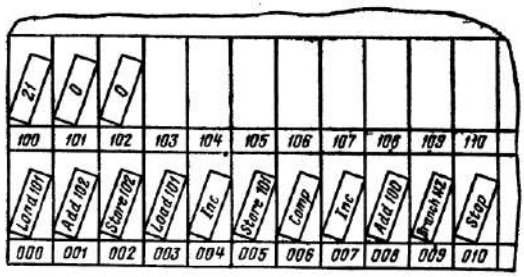


Рис. 11.4. Модель памяти в виде стеллажа

сами. Вместо конкретных чисел принято указывать название символических адресов, поэтому окончательная программа на языке ассемблера выглядела бы так:

Метка	Мнемокод	Символический адрес	Комментарий
START	LDA,	NUM STORE	Вычислить N
	ADD,	SUM STORE	Новое S=S+N
	STA,	SUM STORE	Запомнить новое S
	LDA,	NUM STORE	Вычислить N
	INC		Новое N=N+1
	STA,	NUM STORE	Запомнить новое N
	COMP		} Дополнительный код N
	INC		
	ADD,	KEY STORE	21 — новое N
	BR NZ,	START	Возврат к началу (START)
	BRK		Стоп

Как перевести программу на языке ассемблера в программу в шестнадцатеричных кодах?

Для этого прежде всего потребуется список шестнадцатеричных кодов операций воображаемого микропроцессора:

Мнемокод	Код операции	
LDA	AD	** **
ADD	6D	** **
STA	8D	** **
BR NZ	2B	** **

Это трехбайтные команды, где ** ** — адрес в памяти, откуда выбираются или где хранятся данные.

К однобайтным командам относятся следующие:

Мнемокод	Код операции
INC	16
COMP	17
BRK	00

Теперь программу можно представить в шестнадцатеричных кодах. Начиная с первой команды, можно проследить всю программу, назначая каждой команде ячейку в памяти. Если код операции для первой команды помещен в ячейку 0000_{16}, то, поскольку эта команда занимает 3 байта, код операции для следующей команды должен быть помещен в ячейку 0003_{16}. Разместив таким образом все команды, определяя их местоположение в памяти (в ячейках с шестнадцатеричным адресом) с учетом длины команды, получим следующее:

Адрес	Метка	Мнемокод	Символический адрес	
0000	START	LDA,	NUM	STORE
0003		ADD,	SUM	STORE
0006		STA,	SUM	STORE
0009		LCA,	NUM	STORE
000C		INC		
000D		STA,	NUM	STORE
0010		COMP		
0011		INC		
0012		ADD,	KEY	STORE
0015		BR NZ,	START	
0018		BRK		

Каждому символическому адресу необходимо поставить в соответствие реальный шестнадцатеричный адрес. Очевидно, для символического адреса START (предпоследняя строка) такой адрес уже определен: 0000_{16}. Для остальных: KEY STORE, SUM STORE и NUM STORE шестнадцатеричным адресом может быть любой, бóльший чем 0018_{16} (т. е. любой, до сих пор не использованный, адрес). Пусть, например, их адресами будут соответственно: 0064_{16}, 0065_{16} и 0066_{16}. Тогда получим:

KEY	STORE	←→ 0064
NUM	STORE	←→ 0065
START		←→ 0000
SUM	STORE	←→ 0066

Остается снова просмотреть программу и представить каждую команду (код операции и адрес, если требуется) в шестнадцатеричном коде (с учетом списка шестнад-

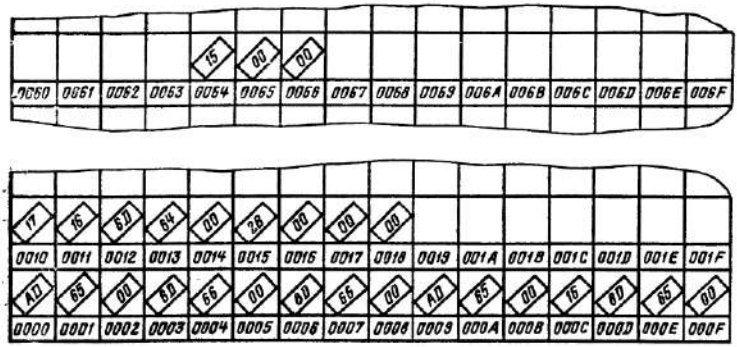


Рис. 11.5. Модель памяти с шестнадцатеричными кодами в виде стеллажа

цатеричных кодов операций для данного микропроцессора) против каждой ячейки. Если в данную микро-ЭВМ вводятся только двоичные коды, придется воспользоваться двоичным представлением вместо шестнадцатеричного.

Таким образом, программа примет следующий вид:

Адрес	Содержимое памяти
0000	AD65 00
0003	6D66 00
0006	8D66 00
0009	AD65 00
000C	16
000D	8D65 00
0010	17
0011	16
0012	6D64 00
0015	2B00 00
0018	00

Первое значение N(0) должно храниться в ячейке 0065_{16}, первое значение S (также 0) — в ячейке 0066_{16}, а число 21_{10}—в ячейке 0064_{16}:

0064 ⟷ 15
0065 ⟷ 00
0066 ⟷ 00

Заметим, что число 21_{10} соответствует 15_{16}. Окончательное размещение программы и данных в памяти указано на рис. 11.5.

Как осуществляется прогон этой программы и что при этом происходит внутри микропроцессора?

Сначала счетчик команд устанавливается в состояние, соответствующее номеру ячейки, в которой размещена первая команда (0000_{16}). При прогоне программы из этой ячейки выбирается код операции первой команды AD_{16} со следующим за ним адресом (сначала самый младший байт) из ячеек 0002_{16} и 0003_{16} по мере того, как на это укажет счетчик команд. После выполнения первой команды начальное значение N(0), которое хранится в ячейке памяти 0065_{16}, загружается в аккумулятор.

Далее аналогичным образом делается выборка следующего кода операции $6D_{16}$ и адреса 0066_{16}, и после выполнения этой команды начальное значение S(0), которое хранится в ячейке памяти 0066_{16}, прибавляется к числу N, содержащемуся в аккумуляторе.

Результатом выполнения первых двух команд S+N (все еще нуль в данный момент) является новая сумма S, которая и будет теперь находиться в аккумуляторе. Далее вызывается следующая команда 8D6600, после выполнения которой новая сумма S посылается на хранение обратно в ячейку 0066_{16} на место прежнего значения S. Затем вызывается очередная команда AD 6500, и после ее выполнения начальное значение N, пока еще хранящееся в ячейке 0065_{16}, переписывается в аккумулятор на место находящегося там числа.

Следующая вызываемая команда INC содержимого аккумулятора с кодом операции 16_{16} выполняет процедуру приращения 1 к числу N. Аккумулятор теперь содержит второе значение N (т. е. N+1), называемое новым N. Когда выбирается очередная команда 8D 6500, то после ее выполнения новое значение N заносится обратно в память на место прежнего содержимого ячейки 0065_{16}.

Аккумулятор все еще сохраняет это новое значение N, поскольку оно не изменилось со времени выполнения команды INC. Выбирается из памяти и выполняется очередная команда с кодом операции 17_{16}, вызывающая поразрядное инвертирование содержимого аккумулятора. Следующая за ней команда 16_{16} прибавляет единицу к содержимому аккумулятора. В аккумуляторе теперь записан дополнительный код второго значения N (обратный код+1 порождает дополнительный код).

По следующей команде 6D 6400 прибавляется число, хранящееся в ячейке 0064_{16}, к содержимому аккумулятора. Это и есть число 21_{10}. С помощью следующей команды 2B 00 00 производится проверка состояния флага нуля, и, поскольку в данном случае результат суммирования отличен от нуля, счетчик команд переводится в состояние, соответствующее номеру ячейки 0000_{16}, в которой содержится код операции первой команды. Так будет каждый раз, пока команда с кодом операции 2B не обнаружит установку флага нуля (т. е. пока новое N будет меньше 21_{10}). Далее вызывается самая первая команда из ячейки 0000_{16}, но вызываемое вслед за ней число N является новым N и в данном случае будет равно 1.

Таким образом, программа в процессе прохождения выполнит несколько повторных переходов по петле с постепенно возрастающими значениями N и S, пока, наконец, новое N не станет равным числу 21_{10}. При этом число, хранимое как сумма S, будет искомым ответом. Условный переход по не нулю не состоится, и счетчик команд перейдет в состояние, соответствующее номеру ячейки 0018_{16}. Выбранная из этой ячейки команда 00_{16} остановит программу, и, если считать содержимое ячейки 0066_{16}, требуемый ответ будет найден.

Этот пример служит всего лишь иллюстрацией того, как можно запрограммировать задачу и каким образом в типовом микропроцессоре будут выполняться операции по реализации написанной программы. К сожалению, «типового микропроцессора» реально не существует, а воображаемый микропроцессор рассматривался с общих позиций с тем, чтобы осветить по возможности наибольшее число особенностей существующих микропроцессоров. Так, например, команда сложения с переносом входит в состав команд далеко не всех микропроцессоров. Для некоторых из них не предусмотрена команда очистки аккумулятора. Большинство микропроцессоров ориентировано на реализацию одной команды вычитания, заменяющей собой три команды (COMP, INC и ADD), использовавшиеся, например, в рассмотренной задаче. При программировании микропроцессора модели 6502 важно помнить, что в составе его команд имеется только команда вычитания с заемом ($A-M-\overline{CF}\rightarrow A$). Если требуется произвести вычитание без заема, необходимо сначала установить флаг переноса. В наборы команд других микропроцессоров входит команда вычитания с перено-

сом (A—M—CF→A) и команда вычитания без переноса (A—M→A).

Для программирования на языке низкого уровня сначала необходимо внимательно ознакомиться с набором команд того микропроцессора, которым вы пользуетесь, чтобы выяснить, какими именно командами он располагает и какой способ адресации характерен для него. Как уже неоднократно упоминалось, состав команд и способы адресации у разных типов микропроцессоров различны.

Сначала число команд микропроцессора может показаться чрезмерно большим, и программы, составленные Вами, будут вероятнее всего не очень эффективными. Но постепенно Вы привыкнете к набору команд своего микропроцессора и убедитесь в том, что каждую программу можно составить несколькими способами. И если при решении какой-либо задачи возникнут затруднения в связи с ограниченными возможностями памяти, придется прибегнуть к наиболее рациональному, с точки зрения экономии памяти, применению имеющихся в вашем распоряжении команд.

Рассмотренному выше микропроцессору был придан очень простой набор команд. В действительности микропроцессоры располагают также более эффективными одиночными командами, дающими возможность выполнять такие операции, как увеличение и уменьшение, а также сдвиг не только содержимого аккумулятора, но и других регистров микропроцессора, а иногда даже и ячеек памяти. Это означает, что при использовании таких команд потребуется реже прибегать к специальному перемещению данных, и программу можно будет записать короче.

Например, если воспользоваться командами очистки аккумулятора (CLA) и уменьшения содержимого ячейки памяти на единицу (DEC) M—1→M, то при условии, что результат действия команды DEC оказывает влияние на флаг нуля,

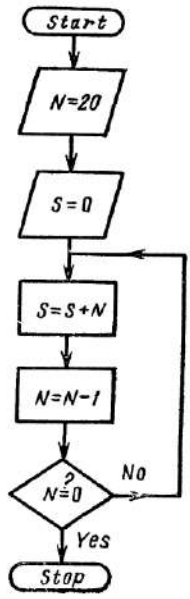


Рис. 11.6. Схема алгоритма сложения, соответствующая более короткой программе

программу сложения целых чисел от 0 до 20 можно записать следующим образом:

Метка	Мнемокод	Символический адрес	Комментарий
START	CLA		S=0
LOOP	ADD	NUM STORE	Прибавить N
	DEC	NUM STORE	Новое N
	BR NZ	LOOP	Переход, если новое N не равно нулю
	STA	ANSWER	Запомнить ответ
	BRK		СТОП

Число 20 помещается в ячейку с символическим адресом NUM STORE, после чего можно запускать программу. Ответ будет содержаться в ячейке памяти ANSWER.

Схема этой программы приведена на рис. 11.6.

Обычно микропроцессоры допускают использование нескольких различных видов адресации, и при написании программы на языке ассеблера необходимо как-то различать данные, используемые сразу же, и адреса памяти, хранящие данные для последующего использования. Один из способов их различения — это применение специального символа # для обозначения немедленно используемых данных. Например, запись LDA#NUM означала бы загрузку действительного значения NUM, а запись LDA, NUM STORE означала бы загрузку числа, хранящегося в памяти по адресу NUM STORE. В другом распространенном способе предполагается использование скобок. В этом случае запись LDA, NUM означает загрузку числа NUM, а запись LDA (NUM STORE) — загрузку числа, хранящегося в памяти по адресу NUM STORE.

ПРИЛОЖЕНИЕ 1

Словарь терминов и сокращений

Адаптер внешнего интерфейса (PIA — Peripheral Interface Adapter) — устройство сопряжения микропроцессора с внешними устройствами.

Адаптер последовательного интерфейса асинхронной передачи данных (ACIA — Asynhronous Communication Interface Adapter) — устройство передачи данных между микро-ЭВМ и внешними устройствами.

Адрес (Address) — 16-разрядное слово, используемое для указания определенного места, обычно ячейки памяти.

Аккумулятор (Accumulator) — специальный регистр микропроцессора, который при выполнении арифметических и логических операций служит источником одного из операндов и местом запоминания результата выполнения операции.

Акустическое устройство связи (Acoustic coupler) — устройство, позволяющее осуществить акустическую связь, не прибегая к электрическим соединениям, например, между МОДЕМом и телефонной линией.

АЛГОЛ (ALGOL — ALGOrithmic Language) — алгоритмический язык, относящийся к языкам программирования высокого уровня.

Алгоритм (Algorithm) — упорядоченный набор действий для решения задач с конечным числом операций, приводящий к детерминирующему ответу.

АЛУ (ALU — Arithmetic and Logis Unit) — арифметическо-логическое устройство.

Аппаратные средства (Hardware) — физически реализованные в виде соответствующей аппаратуры электронные и механические устройства ЭВМ.

Арифметическо-логическое устройство (АЛУ) — основное устройство микропроцессора для выполнения арифметических и логических операций.

Архитектура (Architecture) — любая конструкция или размещение соединенных между собой элементов, отражающая структуру связей и влияющая на основные характеристики ЭВМ.

Асинхронный (Asynchronous) **порядок выполнения операций** — способ, при котором одна завершающаяся операция инициирует начало выполнения следующей, независимо от остальных операций, т. е. несинхронизированный способ выполнения операций.

Ассемблер (Assembler) — служебная программа, преобразующая исходную программу, написанную на языке мнемокодов и символических адресов, в программу в двоичных кодах (объектную программу). В процессе ассемблирования формируется список синтаксических ошибок, содержащихся в исходной программе, и выполняется распечатка исходной и объектной версий программы.

Байт (Byte) — наименьшая адресуемая единица информации — восемь двоичных разрядов.

БЕЙСИК (BASIC — Beginner All-purpose Symbolic Instruction Code) — язык программирования высокого уровня. Предложен как универсальный символический язык для решения задач в режиме диалога человека с машиной. В настоящее время разработана расширенная версия этого языка — БЕЙСИК-плюс.

Библиотека (Library) — набор стандартных программ, составленных для определенного микропроцессора или микро-ЭВМ.

Библиотечная программа (Library routine) — отлаженная программа, содержащаяся в библиотеке.

Биполярный транзистор (Bipolar transistor) — обычный транзистор, в котором управляющий ток проходит через полупроводниковые области *n* и *p*.

Бит (Bit — BInary digiT) — двоичная цифра 0 или 1, или двоичный разряд.

Блок-схема (Flow chart) — графическое представление в форме взаимосвязанных блоков логической последовательности процедур решения задач (схема алгоритма) или выполнения фрагментов программы решения задачи на ЭВМ (схема программы).

Блочная организация памяти (Memory banking) — способ увеличения объема памяти ОЗУ микро-ЭВМ путем использования нескольких блоков по 64 Кбайт каждый, подключаемых по одному с помощью специальных сигналов, отпирающих входы соответствующих микросхем.

Бод (Baud) — единица скорости передачи по каналам связи двоичной информации последовательным кодом; 1 бод равен одному двоичному разряду, передаваемому в секунду.

Большая степень интеграции (LSI — Large Scale Integration) — число компонентов микросхемы свыше 100.

Буквенно-цифровое обозначение (Alphanumeric) — алфавит в широком смысле слова, т. е. набор, включающий буквы, цифры и другие специальные знаки.

Буфер (Buffer) — схема, обеспечивающая прохождение данных между устройствами и шиной данных, а также электрическую изоляцию между отдельными частями системы или согласование их импедансов (полных электрических сопротивлений).

Ввод-вывод (I/O — Input/Output).

Вектор (Vector) — адрес памяти, отсылающий микропроцессор к новой области памяти.

Вектор прерываний (Interrupt vector) — 3-разрядный двоичный код, используемый центральным процессором для формирования начального адреса одной из восьми возможных подпрограмм обслуживания запроса на прерывание.

Восьмеричная (Octal) **система счисления** — система счисления с основанием 8.

Время доступа (Access time) — время, затрачиваемое на обращение к памяти и передачу в микропроцессор одного байта информации.

Вспомогательное запоминающее устройство (Backing store) — запоминающее устройство с большим объемом хранимой информации, используемое дополнительно к основной памяти (кассеты с магнитной лентой, флоппи-диски).

Выбор микросхемы памяти (Selection of the chip) — указание с помощью части адреса памяти, в каком модуле (микросхеме) размещено требуемое слово.

Выборка — машинный цикл, в течение которого выполняется обращение к памяти и передача в микропроцессор байта, хранимого в ячейке памяти.

Высокоимпедансное состояние — одно из состояний выхода тристабильного буфера (не 0 и не 1), при котором выход отключен от нагрузки.

Выход «Video» — специальный выход для подключения бытового телевизора.

Генератор тактовых импульсов (Clock) — (см. **синхрогенератор**).

Группа пользователей (User group) — группа лиц, работающих с однотипными или с идентичными микро-ЭВМ (либо с аппаратурой, в состав которой входят идентичные микропроцессоры).

Данные (Data) — факты или необработанные сообщения, закодированные в двоичной форме (поступившие в микро-ЭВМ наборы 1 и 0). Данные еще не являются собственно информацией (см. **информация**).

Датчик времени (Real-time clock) — устройство, прерывающее микропроцессор в фиксированные интервалы времени, чтобы синхронизировать его работу с событиями, происходящими во внешнем окружении.

Двоичная система счисления (Binary) — система счисления с основанием 2, использующая только две цифры, 0 и 1.

Двоично-десятичное кодирование (BCD — Binary Coded Decimal) — способ представления десятичных чисел, в котором каждая десятичная цифра представлена ее двоичным эквивалентом.

Двоичный разряд (Bit) — разряд в двоичной системе счисления.

Двухрядное расположение выводов (DIL — Dual-In-Line).

Динамическая память (Dynamic memory) — оперативное запоминающее устройство, нуждающееся в регенерации с циклом в несколько миллисекунд.

Диск (Disc) (см. **флоппи-диск**).

Дисплей (VDU — Visual Display Unit) — устройство отображения визуальной информации на базе электронно-лучевой трубки (экрана телевизора).

Длина слова (Word length) — число двоичных цифр, воспринимаемых как слово. Микропроцессоры первого поколения использовали 4-разрядные слова, второго поколения 8-разрядные, а современные микропроцессоры 16-разрядные слова.

Дополнительный код — представление отрицательных чисел дополнением до двух. Получается из **обратного кода** прибавлением единицы в младшем двоичном разряде.

ДОС — дисковая операционная система (DOS — Disk Operating System) — управляющая программа обработки данных и обслуживания при хранении данных в виде файлов на флоппи-дисках.

Драйвер (Driver) (см. **программный драйвер** и **шинный формирователь**).

Дуплексный (Duplex) **канал** — канал передачи данных между двумя устройствами одновременно в двух направлениях.

Естественный язык (Natural language) — язык человеческого общения, например английский.

Задание (Job) — набор задач, подлежащих решению.

Индексная адресация (Indexed addressing) — разновидность относительной адресации, при которой действительный адрес определяется путем сложения содержимого специального индексного регистра с адресом, следующим за кодом операции.

Инициализация (Initialization) — установка всех регистров, флагов и т. п. в начальное (исходное) состояние.

Интегральная микросхема (Integrated circuit).

Интерфейс (Interface) — устройства, управляющие потоком и форматами данных между микропроцессором и внешними устройствами.

Информация (Information) — смысл, придаваемый людьми данным.

ИС (IC) (см. интегральная микросхема).

Исполнительная программа (Executive routine) — главная программа, управляющая выполнением других программ, например программа-монитор.

Исполнительный цикл (Execution cycle) — время выполнения команды, обычно включающее в себя несколько временных циклов (Т-состояний).

Исходная программа (Source program) — программа работы микро-ЭВМ, написанная на символическом языке.

Исходный язык (Source language) — язык исходной программы, отличный от машинного языка.

Карта памяти (Memory map) — схема распределения памяти системы.

Кило (Kilo) — в обычном употреблении означает тысячу, однако в вычислительной технике часто используется для обозначения $2^{10}=1024$ (1 Кбайт = 1024 байт).

Клавиатура (QWERTY) — пульт типа пишущей машинки, названием которого служат первые шесть букв верхнего ряда клавиатуры.

Клавишная панель (Keyboard) — панель, содержащая ключи и кнопки, используемые для ввода в микро-ЭВМ программ и данных.

КМОП-структура (CMOS — Complementary Metal Oxide Semiconductor) — комплементарная МОП-структура или структура, состоящая из пары дополняющих МОП-транзисторов с каналами разного типа.

КОБАЛ (COBAL — COmmon Algorithmic Language) — структурированный язык программирования высокого уровня.

КОБОЛ (COBOL — COmmon Business Oriented Language) — язык программирования высокого уровня.

Код (Code) — набор правил, раскрывающих способ представления данных. Часто используется в значении «язык». Например, «в машинном коде» означает: «на машинном языке».

Код ASCII (American Standard Code for Information Interchange) — американский стандартный код для обмена информацией, т. е. двоичный код, используемый для представления букв, цифр и специальных знаков.

Код EBCDIC — 8-разрядный код данных для тех же самых линий, что и в коде ASCII.

Код операции (Op-code) — комбинация двоичных знаков, указывающая на определенную машинную операцию.

Команда (Instruction) — набор двоичных знаков, воспринимаемый микропроцессором при выполнении определенной операции. Обычная команда состоит либо из кода операции, либо из кода операции плюс данные, либо из кода операции плюс адрес данных.

Контрольная сумма (Check sum) — сумма, используемая в методе проверки на наличие ошибок при передаче данных, при котором

восемь младших двоичных разрядов суммы группы слов инвертируются и посылаются как одно слово вслед за этой группой. По получении контрольная сумма снова вычисляется и прибавляется к переданному инвертированному значению контрольной суммы. При условии отсутствия ошибки в передаче ответ всегда будет $1111\ 1111_2$.

КОРАЛ (CORAL — Computer On-line Real Application Language) — язык высокого уровня, представляющий собой развитие языка АЛГОЛ.

Косвенная адресация (Indirect addressing) — способ адресации, при котором в команде содержится адрес ячейки, где можно найти адрес операнда.

Кросс-ассемблер (Cross-assembler) — программа, дающая возможность микропроцессору синтезировать программу на машинном языке для микропроцессора с иным набором команд.

Курсор (Cursor) — движущийся знак, воспроизводимый на экране дисплея, для указания места, где будет высвечиваться очередной вводимый символ.

Логический элемент (gate) — логическая схема, значение выходного сигнала которой зависит только от значений сигналов на ее входах. Основной элемент электронных схем ЭВМ.

Макрокоманда (Macro instruction) — команда на входном языке, эквивалентная определенному числу команд на машинном языке.

Малая степень интеграции (SSI — Small Scale Integration) — число компонентов интегральной микросхемы (до 10).

Машинный язык (Machine language) — язык, непосредственно используемый микропроцессором. Программа на машинном языке не требует интерпретации.

Мега (Mega) — в обычном употреблении означает миллион, однако в вычислительной технике часто используется для обозначения величины $2^{20}=1\,048\,576$. Например, мегабайт (MB — Megabyte) соответствует 1 048 576 байт.

Мегагерц (MHz — Mega Hertz) — 1 млн. циклов/с.

Место останова программы (Breakpoint) — адрес программы, на котором происходит останов для возможности устранения неисправности или ввода данных.

Метка (Label) — один знак или более, используемые для идентификации оператора программы или позиции информации.

Метод кодирования «без возврата к нулю» (NRZ — Non-Return-to-Zero) — метод кодирования цифровой информации на магнитной ленте.

Микропрограмма (Microprogram) — программа, аппаратно реализованная в микропроцессоре, управляющая его работой во время выборки информации из памяти и ее обработки.

Микропроцессор (Microprocessor) — центральный процессор на одной микросхеме, выполненный с использованием технологии изготовления интегральных микросхем с большой степенью интеграции.

Микросхема (Chip) — электронная схема, выполненная на пластине из кремния небольшой площади, заключенная в пластмассовый корпус.

Микроцикл (Micro cycle) — один шаг в микропрограмме процессора.

Микро-ЭВМ (Microcomputer) — вычислительная машина на базе микропроцессора и устройства памяти.

Мини-флоппи (Mini floppy) (см. **флоппи-диск**).

Мини-ЭВМ (Mini computer) — небольшая ЭВМ, более мощная, чем

микро-ЭВМ, выполненная на базе дискретных логических микросхем, возможно без микропроцессора.

Мнемокод (Mnemonic) — слово или последовательность букв, заменяющая полное слово или фразу, удобную для запоминания.

МОДЕМ (MODEM — MOdulator/DEModulator) — устройство для приема и передачи последовательных данных через акустическое устройство, необходимое для связи с телефонной линией, поскольку электрическое соединение с телефонной сетью запрещено.

Модуль памяти — одна или несколько микросхем, образующих функциональную единицу памяти для хранения машинных слов.

Модифицировать программу (Patch) — буквально: «латать» программу, т. е. изменять ее.

Модулятор RF (RF modulator) — устройство, преобразующее сигнал видеовывода в форму, удобную для подключения к бытовому телевизору.

Мультиплексная передача (Multiplexing) — система передачи информации от нескольких устройств по одному каналу.

Набор возможных сервисных программ «Меню» (Menu) — вопросы и возможные варианты ответа, отображаемые на дисплее, например: «Чем желаете заняться?» — 1) играми, 2) расчетами, 3) назначением встреч.

Набор команд (Instruction set) — состав команд для конкретного микропроцессора.

Начальная загрузка (Bootstrap) — загрузка в ЭВМ комплексной управляющей программы путем использования сначала небольшой подпрограммы ПЗУ для обеспечения считывания более сложной программы.

Неисправность (Bug) — сбой в работе аппаратуры микро-ЭВМ либо ошибка программного обеспечения.

Непосредственная адресация (Immediate addressing) — способ адресации, при котором операнд содержится в теле команды. Операнд хранится в ячейке памяти, которая следует непосредственно за ячейкой, хранящей код операции команды.

Несущая плата (Circuit board) — плата для монтажа различных электронных устройств (в основном микросхем), соединения между которыми выполняются с помощью отпечатанных или протравленных медных дорожек.

Неявная адресация (Implied addressing) — способ адресации, при котором в команде содержится код операции, а адрес операнда подразумевается самой командой. Чаще всего таким способом адресуется содержимое аккумулятора.

Обратный код — представление отрицательных чисел дополнениями до единицы. Получается из **прямого кода** инвертированием значения каждого двоичного разряда.

Объединительная плата (Backplane или Motherboard) — печатная плата с размещенной на ней системой шин, в которую вставляются с помощью торцевых разъемов другие платы, несущие на себе отдельные части (устройства) микро-ЭВМ.

Объектная программа — программа на языке машины, получаемая, как правило, в результате трансляции **исходной программы**.

Объектный код (Object code) — команды программы, представленные на машинном языке.

ОЗУ (RAM) (см. **оперативное запоминающее устройство**).

Операнд (Operand) — данные, над которыми выполняется математическая или логическая операция.

Оперативное запоминающее устройство, или память с произвольным доступом (RAM — Random Access Memory) — память, в любую ячейку которой можно записать и из любой ячейки прочесть данные.

Оператор (Statement) — обобщенная команда на языке высокого уровня; несколько команд в строке.

Операционная система (Operating System) — сложная управляющая программа, часто хранящаяся на флоппи-диске.

Организация ввода-вывода по принципу обращения к памяти (Memory mapped input/output) — метод использования средств ввода-вывода в микро-ЭВМ путем адресации портов ввода и вывода так, как если бы они были ячейками памяти.

Организация циклов (Looping) — метод программирования, при котором фрагмент программы (цикл) выполняется неоднократно (много раз).

Основание системы счисления (Radix) — в десятичной системе счисления, например, число 10, в двоичной — число 2.

Открытый коллектор (O/C — Open Collector) — вывод микросхемы, являющийся коллектором выходного транзистора, позволяющий осуществить подключение нескольких устройств, объединенных по выходам, на одну шину (аналогично тристабильным буферам в КМОП-структурах).

Отладка (Debugging) — устранение ошибок.

Относительная адресация (Relative addressing) — способ адресации, при котором действительный адрес операнда определяется путем суммирования адреса, содержащегося в команде, с числом, которое может быть или адресом данной команды, или адресом первой ячейки текущей страницы памяти, или содержимым одного из регистров центрального процессора.

Память (Memory) — устройство для хранения информации.

Память динамическая (см. **динамическая память**).

Память на магнитных доменах (Bubble memory) — магнитная память, не теряющая информацию при отключении питания. По стоимости и быстродействию занимает промежуточное положение между программируемым ПЗУ и памятью на флоппи-дисках.

Память, не сохраняющая информацию при выключении электропитания (Volatile memory).

Память оперативная (см. **ОЗУ**).

Память постоянная (ROM — Read Only Memory) — постоянное запоминающее устройство, предварительно запрограммированное и содержащее неизменяемую информацию. Используется только для считывания информации.

Память, сохраняющая информацию при выключении электропитания (Non-volatile Memory).

Память статическая (см. **статическая память**).

Параллельная (Parallel) **передача данных** — одновременная передача двух и более разрядов.

ПЗУ (ROM) — постоянное запоминающее устройство (см. **память постоянная**).

Перемещение данных (Data relocate) — перемещение данных из одного в какое-то другое место памяти.

Перепрограммируемое ПЗУ (EPROM — Erasable Programmable Read Only Memory) — программируемое ПЗУ с возможностью стирания информации.

Переход (Branch или Jump) — изменение микропроцессором последовательности выполнения команд в программе.

Персональная ЭВМ (Personal computer) — относительно недорогая микро-ЭВМ, предназначенная для личного пользования.

Печатная плата (Card).

ПИЛОТ (PILOT — Programmed Inquiry Learnings Or Teaching) — язык высокого уровня, разработанный для системы программированного обучения (фирмы IBM) с применением вычислительной машины.

ПЛ/1 (PL/1 — Programming Language) — язык программирования высокого уровня.

Плавающая точка (запятая) (Floating point) — один из способов представления дробных чисел в разрядной сетке машины, основанный на экспоненциальном представлении [см. также **фиксированная точка (запятая)**].

Подпрограмма (Subroutine) — фрагмент основной программы, предназначенный для решения частной задачи, который может быть вызван для исполнения на любом шаге выполнения программы.

Поле (Field) — 1) поле знаков — группа связанных знаков, воспринимаемых как одно целое; 2) область на экране дисплея, например изменение поля (с черного на белом фоне на белое на черном фоне).

Полевой транзистор (FET — Field Effect Transistor).

Полубайт (Nibble) — четыре двоичных разряда.

Помещение данных в стек (Push) — операция передачи данных в стек.

Порт (Port) — специальная схема, обычно часть микросхемы, с помощью которой микропроцессор связывается с внешними устройствами.

Последовательная (Serial) **передача данных** — поразрядная передача данных.

Постоянная память (см. **память постоянная**).

Построчнопечатающее устройство (Line printer) — устройство с печатью результата одновременно целой строкой в отличие от обычного устройства, печатающего по одному знаку.

ППЗУ (PROM) — программируемое ПЗУ.

Прерывание (Interrupt) — временное прекращение выполнения текущей программы с целью решения другой задачи по сигналу запроса на прерывание. Прерывание осуществляется с помощью специального механизма, передающего управление особой подпрограмме обслуживания прерывания. Иногда запросам на прерывание присваиваются приоритеты. Немаскируемые запросы на прерывание всегда удовлетворяются немедленно.

Прерывание по вектору (Vectored interrupt) — система прерываний, в которой каждое из прерываний имеет свой особый адрес.

Принцип «первый поступивший обслуживается первым» (FIFO — First-In-First-Out).

Принцип «последний поступивший обслуживается первым» (LIFO — Last-In-First-Out).

Приоритетная программа (Foreground) — главная программа, реализуемая в первую очередь даже при наличии других задач с более низким приоритетом (см. также **фоновая обработка**).

Проверка четности (Parity) — метод, при котором к передаваемым данным добавляется специальный двоичный разряд с целью контроля наличия в них ошибки. При значении этого разряда,

равном 1, число единиц в слове нечетное, а при равном 0—четное

Прогон (Run) **программы** — выполнение программы.

Программа (Program) — упорядоченный список команд или операторов, выполняя которые микро-ЭВМ осуществляет решение задачи.

Программа ассемблера (см. **ассемблер**).

Программа-интерпретатор (Interpreter) — служебная программа, преобразующая каждый оператор языка высокого уровня в последовательность машинных команд и выполняющая эти машинные команды перед тем, как преобразовывать следующий оператор языка высокого уровня, в отличие от **компилятора**, который воспроизводит всю программу целиком на машинном языке перед ее выполнением.

Программа исходная (см. **исходная программа**).

Программа-компилятор (Compiler) — служебная программа, преобразующая операторы языка высокого уровня либо в операторы языка ассемблера, либо в машинные коды с целью последующего прогона на ЭВМ. Компилятор способен заменить каждый фрагмент программы последовательностью команд машинного языка или подпрограммой, не выполняя программу, в отличие от **интерпретатора**.

Программа-монитор (Monitor) — служебная программа, предназначенная для управления микро-ЭВМ в процессе трансляции, тестирования, корректировки и ввода прикладных программ пользователя.

Программа объектная (см. **объектная программа**).

Программа-отладчик — служебная программа, предназначенная для тестирования объектной программы.

Программатор (Programmer) — устройство для программирования электрически программируемого ПЗУ.

Программист (Programmer) — лицо, составляющее программы.

Программно-аппаратное обеспечение (Firmware) — резидентное программное обеспечение, реализованное в микросхемах ПЗУ, программируемых ПЗУ и перепрограммируемых ПЗУ.

Программное обеспечение (Software) — набор действующих программ.

Программный драйвер (Software driver) — последовательность команд, используемых микро-ЭВМ для преобразования формата данных с целью передачи к определенному внешнему устройству и обратно.

Прозрачный (Transparent) — элемент микропроцессора, например, управляющий флаг или регистр, неизвестный программисту или не находящийся под его непосредственным контролем.

Протокол RS 232 — стандартный протокол для последовательного обмена информацией между ЭВМ и внешними устройствами.

Процессор для обработки текста (Wordprocessor) — программное обеспечение или сочетание программного и аппаратного обеспечения для обработки текста. Например, выделения абзацев, выравнивания полей по правому знаку и т. п.

Прямая адресация (Direct addressing) — способ адресации, при котором адрес операнда содержится в теле команды, после кода операции.

Прямой доступ к памяти (DMA — Direct Memory Access) — способ прямого (непосредственного) доступа к памяти, при котором

осуществляется взаимный обмен данными между внешними устройствами и ОЗУ без участия микропроцессора.

Прямой код — обычный двоичный код.

Работа в реальном масштабе времени (Real-time operation) — работа, используемая в случае, когда информацию необходимо обрабатывать в темпе ее поступления. Обработка должна проводиться достаточно быстро с целью своевременной выдачи результатов, требуемых для управления каким-либо процессом или деятельностью.

Разгрузка памяти (Dump) — передача данных на хранение во вспомогательные запоминающие устройства, такие как накопители на магнитной ленте или флоппи-дисках.

Разделение времени (Time sharing) — режим работы вычислительной системы, в котором она как бы решает множество задач для ряда пользователей одновременно. В действительности микропроцессор в любой момент времени работает только на одного пользователя, выполняя частичное решение задач поочередно в некотором порядке. Однако его ответ настолько быстр по сравнению с реакцией пользователя, что последний этого не ощущает.

Распечатка (Hard copy) — выходные данные, записанные на однократно используемом носителе (например, на бумаге).

Регенерация (Refresh) — процесс периодической (каждые несколько миллисекунд) подачи специальных сигналов в динамическое ОЗУ с целью сохранения его содержимого.

Регистр (Register) — ячейка памяти микропроцессора, обычно содержащая одно слово.

Регистр состояния (Status register) — регистр, используемый для хранения информации об определенных условиях, связанных с данными аккумулятора; называется также флаговым регистром.

Редактирование (Editing) — приведение данных к формату, требующемуся для дальнейшей обработки.

Сверхбольшая степень интеграции (VLSI — Very Large Scale Integration) — число компонентов на одном кристалле свыше 1000.

Сверхоперативная память (Scratch pad) — запоминающее устройство с коротким временем доступа, используемое для кратковременного хранения информации (часто промежуточных результатов).

Световое перо (Light pen) — средство ввода графической информации (графиков, чертежей и т. п.) на экран дисплея с помощью специального сенсорного устройства при его контакте с экраном.

Симплексный (Simplex) **канал** — канал передачи данных между двумя устройствами только в одном направлении.

Синтаксис (Syntax) — грамматика языка программирования.

Синхрогенератор (Clock) — центральный генератор тактовых импульсов, синхронизирующий выполнение микропроцессором всех его операций.

Синхронный (Synchronous) **способ выполнения операций** — способ, при котором выполнение операций упорядочено во времени тактовыми импульсами от специального генератора (синхрогенератора).

Система CUTS (Computer Users' Tape System) — стандарт, оговаривающий способ хранения данных на магнитной ленте в кассетном накопителе в виде последовательности модуляций для представления двоичных единиц и нулей.

Слово (Word) — совокупность двоичных разрядов, воспринимаемая

при обработке как единое целое и помещаемая для хранения в одну ячейку памяти.

Состояние аппаратуры (Environment) — состояние всех регистров, флагов и т. п. в любой момент прохождения программы.

Справочные таблицы (Look-up tables) — информация, хранящаяся в памяти, к которой обращается микропроцессор с целью поиска ответа вместо того, чтобы производить вычисления.

Средняя степень интеграции (MST — Medium-Scale Integration) — число компонентов микросхемы от 10 до 100.

Статическая память (Static memory) — оперативное запоминающее устройство, не нуждающееся в непрерывной регенерации, пока включено электропитание, более дорогостоящее по сравнению с динамическим ОЗУ.

Стек (Stack) — область памяти, адресуемая с использованием регистра — **указателя стека.** Обращение к стеку производится в соответствии с принципом **«последний поступивший обслуживается первым».**

Строка (String) — группа элементов данных, хранящаяся в последовательных ячейках памяти и рассматриваемая как отдельная единица информации при программной обработке.

Сумматор с ускоренным переносом (Look ahead) — суммирующая схема арифметическо-логического устройства, формирующая сигналы переноса сразу для всех двоичных разрядов.

Схема алгоритма программы (см. **блок-схема**).

Счетчик (Counter) — регистр, содержимое которого используется для представления числа происходящих событий.

Счетчик команд (Program counter) — регистр, содержащий адрес очередной части программы, подлежащей выполнению.

Тактовый генератор (Clock) (см. **синхрогенератор**).

Терминал (Terminal) — устройство ввода, например, клавиатура; либо устройство вывода, например принтер или дисплей; либо и то и другое вместе.

Тестовая задача (Benchmark) — типовая задача, для решения которой составляются программы для различных микропроцессоров с целью сравнения эффективности их использования.

Технология кремний-на-сапфире (КНС) (SOS—Silicon on Saphire).

Транзистор типа *n*-МОП (NMOS—N-type Metal Oxide Semiconductor) — полевой транзистор с каналом *n*-типа; *n*-МОП-транзистор (металл—окисел—полупроводник).

Транзистор типа *p*-МОП (PMOS—P-type Metal Oxide Semiconductor) — полевой транзистор с каналом *p*-типа; *p*-МОП-транзистор (металл—окисел—полупроводник).

Триггер (Flip-flop) — логический элемент, который может находиться в двух состояниях; используется для хранения двоичных цифр.

Тристабильный выход (TSO—Tri-State Output) — выход с тремя состояниями: 0, 1 и состоянием высокого импеданса.

Т-состояние (T-state) — время одного цикла генератора тактовых импульсов.

Указатель стека (Stack pointer) — регистр, содержимое которого указывает на адрес ячейки, находящейся на дне свободного пространства стека.

Универсальный асинхронный приемопередатчик (UART—Universal Asynchronous Receiver Transmitter) — устройство для асинхронной передачи данных.

Универсальный синхронный/асинхронный приемопередатчик

(USART — Universal Synchronous Asynchronous Receiver Transmitter) — устройство для синхронной или асинхронной передачи данных.

Упаковка в корпус с двухрядным расположением выводов (DIP — Dual-In-line-Package).

Условный переход (Conditional branch или Conditional jump) — изменение микропроцессором последовательности выполнения команд в программе при выполнении или невыполнении некоторого условия.

Файл (File) — набор семантически связанных между собой данных, воспринимаемых как одно целое, хранящихся в форме магнитной записи на флоппи-диске.

Фиксированная точка (запятая) (Fixed point) — один из способов представления дробных чисел в разрядной сетке машины: см. также **плавающая точка (запятая).**

Флаг (Flag) — триггер, который может устанавливаться или сбрасываться по результатам прохождения программ для указания некоторых специфических условий.

Флаговый регистр (Flag register) — регистр, состоящий из нескольких одноразрядных флагов; см. **регистр состояния.**

Флоппи-диск (Floppy disc) — гибкий пластмассовый диск с таким же покрытием из магнитного материала, какое используется для магнитных лент. Флоппи-диски применяются для хранения больших объемов информации, размещенной в концентрических дорожках на их поверхностях.

Фоновая обработка (Backgrounding) — выполнение программы решения задач с низким приоритетом при отсутствии задач с более высоким приоритетом.

ФОРТРАН (FORTRAN — FORmula TRANslation) — язык программирования высокого уровня.

Центральный процессор (Central Processing Unit) — основное устройство любой вычислительной машины, реализующее арифметические и логические операции, а также функции управления.

Цифровая (Digital) ЭВМ — ЭВМ для обработки дискретной информации (в отличие от аналоговой).

ЦП (CPU) (см. **центральный процессор).**

Шестнадцатеричная система счисления (Hexadecimal) — система счисления с основанием 16, использующая цифры от 0 до 9 и шесть букв латинского алфавита A, B, C, D, E, F.

Шина (Bus) — группа линий передачи информации, объединенных общим функциональным признаком, например: шина данных, шина адресов, шина управления.

Шина адресная (Address bus) — система линий передачи адресов.

Шина данных (Data bus) — система линий передачи данных внутри микропроцессора и вне его.

Шина управления (Control bus) — система линий передачи сигналов управления.

Шинный формирователь (Bus driver) — дополнительная микросхема, подключенная к шине данных, выполняющая функции усилителя мощности.

Электрически программируемое ПЗУ (Electrically alterable read only memory) — однократно программируемые ПЗУ после их изготовления.

Электронно-лучевая трубка (Cathode ray tube) — трубка, используемая в качестве экрана в дисплее.

ЭЛТ (CRT) (см. **электронно-лучевая трубка**).

ЭП ПЗУ (EAROM) (см. **электрически программируемое ПЗУ**).

Язык (Language) — система команд, понятных как программисту, так и ЭВМ. Команды на машинном языке различны для разных микропроцессоров, а языки высокого уровня, такие как БЕЙСИК, практически могут использоваться при программировании на любой машине, имеющей соответствующий транслятор с языка высокого уровня.

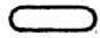
Язык ассемблера (Assembly language) — язык мнемокодов для символического представления предложений программы, удобной при ее написании и последующем переводе ее на язык машинных команд.

Язык высокого уровня (High level language) — язык программирования, которым удобно пользоваться при написании программы решаемой задачи.

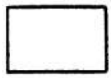
ПРИЛОЖЕНИЕ 2

Обозначения условные графические элементов схем

А. Схемы алгоритмов



Начало и конец алгоритма (начальный элемент снабжается одной исходящей стрелкой, конечный — одной входящей)



Действие или процесс (обработка) (элемент снабжается одной входящей и одной исходящей стрелками)

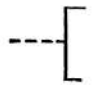
Ввод-вывод данных или результатов (элемент снабжается одной входящей и одной исходящей стрелками)

Проверка условия и принятие решения (элемент снабжается одной входящей и двумя или реже более исходящими стрелками)

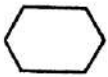
Связующее звено или соединитель (обычно используется для связи частей схемы, размещенных на разных страницах, когда схема занимает более одной страницы. Внутри кружка проставляется номер соединения)

Соединитель различных частей схемы на той же самой странице

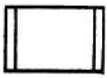
Направленные соединения в схеме



Примечания, дополнения, комментарии



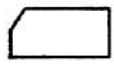
Подготовка или составление данных, процесса, документов



Предварительно определенный процесс, например подпрограмма

Б. Структурные схемы систем

Структурные схемы систем предназначены для того, чтобы проиллюстрировать или провести анализ прохождения данных через систему. Поскольку такая схема должна отражать особенности ввода и вывода данных, в схеме используются символы, форма которых подсказывает способ представления данных, способ их подготовки и способ ввода и вывода:


Данные на перфокартах

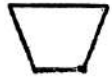

Документы


Ручной ввод


Дисплей


Диск


Магнитная лента


Ручная операция

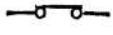
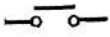
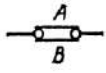
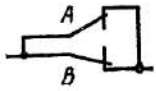

Объединение, например, массивов информации или слива данных


Выделение или извлечение, например, части числа

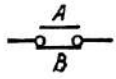
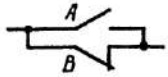
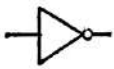
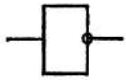
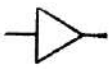
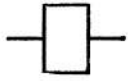
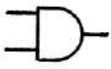
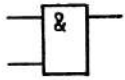
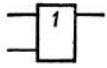
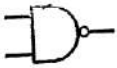
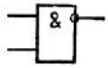
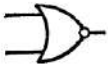
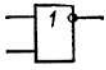
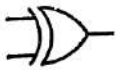
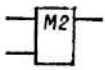

Вспомогательная операция

ПРИЛОЖЕНИЕ 3

Таблица используемых обозначений элементов

У автора	По ГОСТ	Примечание
		Контакт замкнут
		Контакт разомкнут
		Два замкнутых контакта

Продолжение

У автора	По ГОСТ	Примечание
		Два контакта: разомкнутый и замкнутый
		НЕ
		ДА (усилитель)
		И
		ИЛИ
		И-НЕ
		ИЛИ-НЕ
		Сложение по модулю 2

Что дополнительно можно прочитать о микро-ЭВМ и микропроцессорах? [1]

Существует довольно большой объем литературы, в оновном специальной, по указанной тематике как отечественных, так и зарубежных авторов. Ниже приводится перечень некоторых изданий, ко-

[1] Добавлено при переводе. *(Прим. пер.)*

торые могут оказаться весьма полезными при желании более глубоко изучить этот интересный предмет.

1. **Соучек Б.** Микропроцессоры и микро-ЭВМ: Пер. с англ./Под ред. А. И. Петренко. — М.: Советское радио, 1979. — 518 с.

2. **Клингман Э.** Проектирование микропроцессорных систем: Пер. с англ./Под ред. С. Д. Пашкеева. — М.: Мир, 1980. — 568 с.

3. **Вайда Ф., Чакань А.** Микро-ЭВМ: Пер. с венгерск./Под ред. В. В. Сташина. — М.: Энергия, 1980. — 360 с.

4. **Коффрон Дж.** Технические средства микропроцессорных систем. Практический курс: Пер. с англ./Под ред. Л. В. Шабакова. — М.: Мир, 1983. — 344 с.

5. **Поспелов Д. А.** Арифметические основы вычислительных машин дискретного действия. — М.: Высшая школа, 1970. — 308 с.

6. **Программирование** микропроцессоров: Пер. с нем./Под ред. В. В. Сташина. Библиотека по автоматике, вып. 622. — М.: Энергоиздат, 1982. — 85 с.

7. **Бедрековский М. А., Волга В. В., Кручинкин М. С.** Микропроцессоры. — М.: Радио и связь, 1981. — 96 с.

8. **Гивоне Д., Россер Р.** Микропроцессоры и микрокомпьютеры. Вводный курс: Пер. с англ. М.: Мир, 1983. — 460 с.

9. **Дмитриев Ю. К., Хорошевский В. Г.** Вычислительные системы из мини-ЭВМ/Под ред. Э. В. Евреинова. — М.: Радио и связь, 1982.— 304 с.

10. **Микропроцессорные** БИС и микро-ЭВМ. Построение и применение/Под ред. А. А. Васенкова. — М.: Советское радио, 1980. — 280 с.

11. **Микропроцессорные** комплекты интегральных схем: Состав и структура: Справочник/Под ред. А. А. Васенкова, В. А. Шахнова. — М.: Радио и связь, 1982. — 192 с.

12. **Микро-ЭВМ.** Пер. с англ./Под ред. В. В. Сташина. — М.: Энергоиздат, 1982. — 328 с.

13. **Микро-ЭВМ** «Электроника С5» и их применение/Под ред. В. М. Пролейко. — М.: Советское радио, 1980. — 160 с.

14. **Прангишвили И. В.** Микропроцессоры и микро-ЭВМ. — М.: Энергия, 1979. — 232 с.

15. **Уокерли Дж.** Архитектура и программирование микро-ЭВМ. В 2 книгах: Пер. с англ./Под ред. А. Г. Филиппова. — М.: Мир, 1984. — Кн. 1 — 486 с.; кн. 2 — 341 с.

16. **Эдельман Ф. Л.** Структура компонентов БИС. — Новосибирск: Наука, СО АН СССР, 1980. — 256 с.

17. **Алексеенко А. Г., Галицин А. А., Иванников А. Д.** Проектирование радиоэлектронной аппаратуры на микропроцессорах. — М.: Радио и связь, 1984. — 272 с.

Оглавление

70 к.

ЭНЕРГОАТОМИЗДАТ